Timebars Ltd
Agile PM | Classic PM | PPM

Common_12 — Data Migration Assistance Guide

Migrating from a Legacy System to the Timebars Ltd Suite

Table of Contents

---

1. Introduction and Audience

This guide is for the IT staff, data leads, and developers responsible for extracting project data from a legacy system and loading it into Agilebars, Timebars, or Costbars. It describes what data needs to be prepared, which stores it belongs in, how the stores relate to each other, and how to use the Spreadsheet Sync feature as the import mechanism.

Who should read this guide:

- Database administrators or developers writing export scripts from the legacy system
- IT leads coordinating the migration effort
- Project managers overseeing what data needs to migrate and what can be left behind

What this guide does not cover:

- How to operate the Spreadsheet Sync feature day-to-day — for that, see Spreadsheet Sync User Guide
- Full field definitions for every store — for that, see Data Model and Scheduling Engine Guide and MetaData Fields Detail Report

---

2. Migration Mechanism Overview

How the Application Stores Data

The Timebars Ltd suite is a browser-based single-page application. All data lives in **IndexedDB**, an in-browser database — there is no server-side backend or REST API to push data into. The formal and supported path for loading data into the application is the **Spreadsheet Sync** feature.

Spreadsheet Sync accepts a specially structured spreadsheet file (Excel `.xlsm` or LibreOffice `.ods`) that is dragged onto the application canvas. The spreadsheet worksheets map directly to the IndexedDB data stores. For migration, the task is to populate those worksheets with data from the legacy system and then import the spreadsheet.

Two Migration Paths

Path A — CSV Files via the Spreadsheet Template

The customer's IT team exports data from the legacy system as CSV files, with column headers that exactly match the Timebars field names. Those CSV files are loaded into the Timebars spreadsheet template using the Import buttons on each worksheet tab. The populated spreadsheet is then drag-dropped onto the canvas.

This is the recommended path for most migrations. It works well when IT staff can write SQL export queries or use ETL tools, and does not require programming skills beyond generating CSV output.

Path B — Direct JSON Import

Developers write a program that reads from the legacy system and writes data directly in the Timebars JSON format — the same field names and structure documented in the Data Model guide. The resulting JSON file can be drag-dropped directly onto the canvas without using the spreadsheet template as an intermediary.

This path suits large-volume or automated migrations, repeat-load scenarios (e.g. nightly delta sync during a transition period), or teams with developer resources who prefer to work at the data level rather than via spreadsheet.

Choosing a Path:

Consideration	Path A — CSV	Path B — JSON
Developer resources required	Low — SQL or ETL output	Medium — requires a program or script
Volume	Any	Better for very large datasets
Repeat loads	Manual re-import each time	Easily automated
Validation feedback	Via spreadsheet red-highlighting	Via app import dialog
Recommended for	Most migrations	Large or automated migrations

3. Source Tables — What IT Needs to Populate

Of the twelve data stores in the Timebars suite, only **five are source tables** that a migration needs to supply. The remaining stores are either system-generated from these five, or are configuration stores best set up through the application UI after migration.

The five source tables are:

Store	Spreadsheet Tab	Purpose
`tbTimebars`	Timebars	Every scheduling item: Portfolios, Projects, Tasks, Allocations
`tbMetaData`	MetaData	Extended classification, narrative, and scoring fields
`tbBaseline`	Baseline	Approved baseline plan (optional — can be set post-migration)
`tbTags`	Tags	Picklist dropdown values for metadata fields
`tbResources`	Resources	Resource pool — people and generic roles

The field structures for all five stores are documented in full in Data Model and Scheduling Engine Guide.

3a — tbTimebars (Worksheet: Timebars)

`tbTimebars` is the backbone of the application. Every scheduling item at every level of the hierarchy — Portfolio, Project, Sub-Project, Task, Milestone, and Allocation — is a single row in this store. The hierarchy is defined entirely by the `tbSelfKey2` field, which stores the `tbID` of the parent row.

This is the most critical table in the migration. Every other table either points into it or derives from it.

Hierarchy Levels

`tbType` value	Level	Description
`Portfolio` or `Program`	L1	Top-level grouping
`Project`	L2	A discrete deliverable or initiative
`Sub-Project`	L3	Work package or component within a project
`Task` or `Milestone`	L4	Individual work items or schedule control points
`Allocation`	L5	A specific resource assigned to a task

Agilebars note: Agilebars uses only L2 (Project) and L4 (Task). L1, L3, and L5 are not supported in Agilebars.

Critical Fields for Migration

Field	Required	Notes
`tbID`	Yes	Unique string identifier for this row. Can be a legacy system ID converted to a string.
`tbSelfKey2`	Yes (except root)	The `tbID` of this row's parent. Null or empty for Portfolio rows (root nodes). **This single field defines the entire hierarchy.**

Field	Required	Notes
`tbType`	Yes	Must be one of the values in the hierarchy level table above.
`tbName`	Yes	Display name for this item.
`tbStart`	Yes	Forecast start date in `DD-MMM-YYYY` format (e.g. `15-Jan-2026`).
`tbFinish`	Yes	Forecast finish date in `DD-MMM-YYYY` format.
`tbStatus`	Recommended	Typically `"Active"` for live items.

Work and Cost Fields

Enter cost and work values **only at the leaf level** — L5 Allocations, or L4 Tasks when no Allocations are used. Leave these fields as `0` on parent rows (Project, Sub-Project, Portfolio). The rollup engine recalculates parent totals automatically from their children on import.

Field	Description
`tbWork`	Forecast hours
`tbAWork`	Actual hours to date
`tbWorkRemaining`	Remaining hours
`tbCost`	Forecast cost
`tbACost`	Actual cost to date
`tbCostRemaining`	Remaining cost
`tbPercentComplete`	Progress (0–100 integer)

Allocation-Specific Fields (L5 only)

L5 Allocation rows link to the resource pool via `tbResID` and drive the scheduling engine's cost and hour calculations.

Field	Description
`tbResID`	Must match a `tbResID` in the `tbResources` store
`tbCalendar`	Working hours per day for this resource
`tbPercentTimeOn`	Percentage of resource time on this task (0–100)
`tbPayRate`	Hourly pay rate — used to compute cost from hours

Fields to Leave Blank on Import

The following fields are system-generated and will be populated automatically by the application after import. Do not attempt to populate them from the legacy system:

- `tbL1`, `tbL2`, `tbL3`, `tbL4`, `tbL5` — ancestor name breadcrumbs, regenerated on load

- `tbHierarchyOrder` — WBS position string, regenerated on load
- `tbCoordTop`, `tbCoordLeft`, `kbCoordTop`, `kbCoordLeft` — canvas and Kanban board pixel positions, set by the rendering engine. The Timebars worksheet in the spreadsheet template includes a macro button to auto-generate these coordinate values based on the order the rows appear in the sheet. It is therefore important that migration data is arranged in the same hierarchy order as it existed in the legacy system — parent rows above their children, and siblings in the intended display sequence — before running the auto-generate macro.
- `tbDuration`, `tbRemainingDuration` — these are calculated from the bar's start and finish dates after bars are rendered and scheduled on the canvas. Either enter `0` for both fields on import, or pre-calculate them as working-day counts if your export script can do so. They will be recalculated correctly by the scheduling engine on first use.
- `tbCostID`, `tbStatus`, `tbState`, `tbStep`, `tbStepStatus`, `tbWfReasonNote`, `tbStage`, `tbPriority`, `tbPhase`, `tbBarColor`, `tbTextColor`, `focdItemCoordLeft`, `focdItemCoordTop`, `tbBarHeight`, `tbBLID`, `tbConstraintType`, `tbConstraintDate`, `tbFloat`, `tbFreeFloat` — these fields are not used by the core system and can be left blank or null on import.
- `tbCustomerID` — can be left blank, or set to any string value representing the tenant or organisational unit the item belongs to (e.g. `"EngDept"`, `"MarketingDept"`). When set, it acts as a partition identifier that separates items by owner for the enterprise dashboard. Leave blank if tenant separation is not required.
- `canvasNo` — always set this to `1`.

3b — tbMetaData (Worksheet: MetaData)

`tbMetaData` holds over 100 extended fields for classification, narrative text, financial scoring, health indicators, risk and issue tracking, and portfolio assessment. It is joined to `tbTimebars` by a matching ID: the `tbMDID` in this store must equal the `tbID` of the corresponding `tbTimebars` row.

One `tbMetaData` row must exist for every `tbTimebars` row. The application expects this one-to-one relationship. For items with no metadata in the legacy system, create a row with `tbMDID` set to the matching `tbID` and all other fields left null.

Key Fields

Field	Notes
`tbMDID`	Must match the `tbID` of the corresponding `tbTimebars` row
`tbMDName`	Typically auto-set to match `tbName` — provide the same value
`tbMDStatus`	Item status — value must exist in `tbTags` under group "Status"
`tbMDPriority`	Priority — value must exist in `tbTags` under group "Priority"
`tbMDPM`	Project manager name
`tbMDBusinessOwner`	Business owner

Field	Notes
tbMDSponsoringDepartment	Sponsoring department
tbMDNotesProject	General project description or notes (rich text)
tbMDHealthOverall	Overall health indicator
tbMDCategory	Project or item category

Mapping Legacy Fields to tbMetaData

The full field listing is in MetaData Fields Detail Report. Common mappings from typical legacy systems:

Legacy System Field	Timebars MetaData Field
Status / State	tbMDStatus
Priority / Urgency	tbMDPriority
Project Manager	tbMDPM
Business Owner	tbMDBusinessOwner
Department	tbMDSponsoringDepartment
Description / Scope	tbMDObjectivesAndScope
Notes / Comments	tbMDNotesProject
Project Number / Reference	tbMDProjectNumber
Health / RAG Status	tbMDHealthOverall
External Link / URL	tbMDExtLink1
External System ID	tbMDExtSystemID1
Azure DevOps ID	tbMDAzureID
Risk Score	tbMDScore
Risk Probability	tbMDProbability
Risk Impact	tbMDImpact

Picklist fields (Status, Priority, Health, Category, Phase, etc.) must reference values that exist in tbTags. Load tbTags first and confirm the picklist values are present before importing tbMetaData.

3c — tbBaseline (Worksheet: Baseline)

tbBaseline stores a point-in-time snapshot of the approved schedule — the plan-of-record against which progress is measured. Its field structure is identical to tbTimebars. Refer to the tbTimebars field reference in Data Model and Scheduling Engine Guide for all field names.

Baseline is optional for migration. Two approaches:

- **If the legacy system holds an approved baseline plan:** Populate `tbBaseline` with those approved dates and hours using the same field structure as `tbTimebars`. This gives users an immediate baseline comparison on day one.
- **If no baseline exists in the legacy system:** Skip this table. After migration, set the baseline from within the application (Right-click Canvas > Set Baseline) once the imported schedule has been reviewed and approved.

The Baseline worksheet in the SS template is normally marked as a system-generated store. For migration only, it is acceptable to pre-populate it following the `tbTimebars` structure.

3d — tbTags (Worksheet: Tags)

`tbTags` defines all picklist dropdown values used in the metadata forms. Each row is a single option within a named group. The `tbTagGroup` identifies which picklist the option belongs to, and `tbTagTbInternalName` identifies which `tbMetaData` field it populates.

Load tags before MetaData. If a `tbMetaData` row references a picklist value that does not exist in `tbTags`, the import will flag a validation error.

Key Fields

Field	Description
`tbTagID`	Unique identifier — must be unique across all tag rows
`tbTagGroup`	The picklist group name (e.g. "Status", "Priority", "Phase")
`tbTagName`	The option value displayed to users
`tbTagNameShort`	Abbreviated label for compact display
`tbTagTbInternalName`	The `tbMetaData` field this group populates (e.g. `tbMDStatus`)
`tbTagPurpose`	Description of the tag group's purpose

Migration Approach for Tags

The application ships with a full set of seed tag data covering common picklists (Status, Priority, Health, Phase, Category, Investment Type, Benefit Cost Ratio, etc.). The migration team should:

1. Review the existing seed tags in the Tags worksheet.
2. Identify any customer-specific values in the legacy system that are not already covered.
3. Add those customer values as new rows in `tbTags` with the appropriate `tbTagGroup` and `tbTagTbInternalName`.
4. Remove or rename seed values that do not apply to the customer's terminology (optional but recommended for a clean setup).

Tags are typically a small dataset — most migration teams build this table manually by reviewing the legacy system's dropdown lists rather than writing an export query.

3e — tbResources (Worksheet: Resources)

`tbResources` holds the resource pool. Each row represents one named person (Human) or a generic role placeholder (Generic). L5 Allocation rows in `tbTimebars` reference a resource via `tbResID`.

Load resources before Allocations. Allocations that reference a `tbResID` that does not exist in `tbResources` will fail validation on import.

Key Fields for Migration

Field	Description
`tbResID`	Unique resource identifier — referenced by `tbResID` on L5 Allocation rows
`tbResName`	Full display name
`tbResPayRate`	Hourly cost rate used by the scheduling engine
`tbResResourceCalendar`	Working hours per day (typically `"8"`)
`tbResPercentGeneralAvailability`	Percentage of working time generally available (0–100)
`tbResLabourType`	`"Human"` or `"Generic"`
`tbResPrimaryRole`	Role category — used in resource reports and stacked charts
`tbResPrimarySkill`	Primary skill area
`tbResDepartment`	Organisational department
`tbResManager`	Manager name
`tbResLocation`	Office or location
`tbResPartTimeFullTime`	`"Full Time"` or `"Part Time"`
`tbResResourceType`	`"Labour"` or `"Non-Labour"`
`tbResResourceClass`	Employment class (e.g. `"Permenant"`, `"Contract"`)
`tbResCostCode`	Charge or cost code for financial integration
`tbResMonth1–Month24`	Monthly availability factors (0.00–1.00) over a 24-month window
`tbResDaysNotAvailableByMonth`	Exceptions: comma-separated month+year and day-count (e.g. `"Jan25-2, Feb25-2"`)
`tbResExtSystemResID`	Carry-over ID from the legacy system for traceability

The full `tbResources` JSON structure is documented in Data Model and Scheduling Engine Guide.

---

4. Pre-Processed and System-Generated Tables — Do Not Populate

The following stores are computed from the five source tables. Do not attempt to populate them from the legacy system — they will be rebuilt automatically by the application after import.

Store	Generated From	When Rebuilt
`tbMdJoined`	Join of tbTimebars + tbMetaData	On every load and recalculate
`tbResCalcs`	Allocation data in tbTimebars	After Recalculate All
`tbResCalcs2`	Allocation data + tbResources	After Recalculate All

The following stores are configuration, not migration data:

Store	Purpose	How to Set Up
`tbAdminPanel`	Canvas settings, status/report date, holidays, product configuration	Via the Admin Panel within the application after migration
`tbFields`	Dynamic form field definitions	Ships with defaults; customise via the application or SS Sync post-migration
`tbCoreReport`	Metadata field visibility per hierarchy level	Ships with defaults; adjust via the application after migration

5. ID Strategy and Hierarchy Wiring

The most common migration failure is broken hierarchy wiring — either IDs that do not match across tables, or `tbSelfKey2` values that point to nonexistent `tbID` values.

ID Options

Option 1 — Carry over legacy system IDs. If the legacy system uses unique, stable identifiers (project numbers, task IDs, work package codes), convert them to strings and use them as `tbID` values. Set `tbResExtSystemResID` on resource rows and `tbMDExtSystemID1` on metadata rows to preserve the legacy ID for traceability. This makes post-migration reconciliation easier.

Option 2 — Generate new IDs. If legacy IDs are not unique, are numeric in a format that may collide across tables, or are otherwise unreliable, generate clean new IDs for the migration. Sequential integers, timestamps, or prefixed strings all work. The application has no constraint on ID format beyond uniqueness.

Whichever option is used: **every value that appears in a `tbSelfKey2` field must exist as a `tbID` somewhere in the same `tbTimebars` dataset.** This invariant must hold before import.

Example Hierarchy Chain

The following illustrates how a small hierarchy of five rows wires together through `tbID` and `tbSelfKey2`:

tbID	tbName	tbType	tbSelfKey2
P001	Digital Workplace	Portfolio	*(empty — root node)*
P002	Cloud Migration	Project	P001

tbID	tbName	tbType	tbSelfKey2
P003	Phase 1 — Assessment	Sub-Project	P002
P004	Vendor Assessment	Task	P003
P005	Alloc — Jane Smith	Allocation	P004

The `tbMetaData` table then has one row for each of P001 through P005, with `tbMDID` matching each `tbID`. The `tbResources` table has a row for Jane Smith, and the Allocation row P005 carries Jane Smith's `tbResID`.

Pre-Import Validation Check

Before loading the data, validate the hierarchy wiring:

- Every non-empty `tbSelfKey2` value in `tbTimebars` must exist as a `tbID` in `tbTimebars`.
- Every `tbMDID` in `tbMetaData` must match a `tbID` in `tbTimebars`.
- Every `tbResID` on an Allocation row in `tbTimebars` must exist in `tbResources`.
- All `tbTagName` values used in `tbMetaData` picklist fields must exist in `tbTags` under the correct `tbTagGroup`.

These four checks will catch the vast majority of referential integrity issues before the import reaches the application.

---

6. Date and Numeric Format Rules

These rules are strict. The import engine does not attempt to parse alternative formats.

Dates

- Format: `DD-MMM-YYYY` — for example, `15-Jan-2026`
- Month abbreviation must be exactly three letters with initial capital: `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov`, `Dec`
- Do not use ISO 8601 (`2026-01-15`), US format (`01/15/2026`), or any other variant
- Enforce this format in the export script or ETL transformation

Numbers

- Numeric fields must be numbers, not quoted strings
- Default value for numeric fields with no data: `0` — never `null` or empty string
- `tbPercentComplete`: integer 0–100, not a decimal fraction (0.0–1.0)
- Work and cost values: decimal numbers are acceptable (`40.5` hours)
- Pay rates: decimal numbers (`75.00`)

Text Fields

- Text fields with no value: empty string `""`, not `null`
- Exception: fields documented as accepting `null` in the JSON reference (such as most `tbMetaData` fields) may carry `null` when the value is genuinely absent

Boolean-style Fields

- The spreadsheet stores these as `"Yes"` / `"No"` text strings
- In JSON format, use the string values `"Yes"` and `"No"`, not true/false booleans

---

7. Step-by-Step Migration Process

Path A — CSV Files via the Spreadsheet Template

Preparation

1. Download the Timebars spreadsheet template: Excel (`.xlsm`) or LibreOffice Calc (`.ods`) from the Hamburger Icon > Download menu.
2. Do not rename the file — it must begin with `tbClient` for the import to work.
3. Open the template and configure the Setup Tab: set the Location column to the folder where you will place the CSV files.

Exporting from the Legacy System

4. IT staff export data from the legacy system as CSV files.
5. Name each file to match the Timebars store name exactly: `tbTimebars.csv`, `tbMetaData.csv`, `tbBaseline.csv`, `tbTags.csv`, `tbResources.csv`.
6. Column headers in each CSV must exactly match the Timebars field names as documented in the Data Model guide. Case is significant.
7. Place all CSV files in the folder configured in Step 3.

Loading into the Spreadsheet

8. In the spreadsheet, navigate to the **Setup Tab**.
9. Confirm the "Import Yes/No" column is set to Yes for each of the five source stores.
10. Click **Import All** to load all five CSVs simultaneously, or use the individual Import button on each worksheet tab.
11. Review each worksheet tab for red-highlighted cells — these indicate validation issues that must be resolved in the source data before proceeding.
12. Fix any issues in the export scripts, regenerate the CSV files, and re-import.

Importing into the Application

13. Save the spreadsheet file (keeping the `tbClient` filename prefix).
14. Open the Timebars application in the browser.
15. Drag and drop the saved spreadsheet file onto the Canvas.
16. The application creates an automatic backup before processing.
17. IndexedDB is updated and the Canvas refreshes.

Post-Import

18. Run **Recalculate All** (Right-click Canvas > Recalculate All) to rebuild hierarchy labels, rollup totals, and resource calculations.
19. Proceed to the Validation Checklist in Section 8.

Path B — JSON Direct Import

Preparation

1. Review the JSON field structures in Data Model and Scheduling Engine Guide for all five source stores.
2. Developers write a program or script that reads from the legacy system and produces JSON in the exact Timebars format, with all five stores represented.

Import

3. The JSON output file can be drag-dropped directly onto the Canvas, or imported via Hamburger Icon > File Importer.
4. The application creates an automatic backup before processing.
5. Run **Recalculate All** after import.
6. Proceed to the Validation Checklist in Section 8.

---

8. Validation Checklist

Pre-Import Checks (in the source data or spreadsheet)

- ☐ Every `tbSelfKey2` value in `tbTimebars` resolves to a `tbID` in the same dataset
- ☐ No duplicate `tbID` values in `tbTimebars`
- ☐ Every `tbMDID` in `tbMetaData` matches a `tbID` in `tbTimebars`
- ☐ One `tbMetaData` row exists for every `tbTimebars` row
- ☐ Every `tbResID` on an Allocation row in `tbTimebars` exists in `tbResources`
- ☐ All picklist values in `tbMetaData` exist in `tbTags` under the correct group
- ☐ All dates are in `DD-MMM-YYYY` format
- ☐ All numeric fields contain numbers, not strings or nulls
- ☐ No red-highlighted cells remain in the spreadsheet template

Post-Import Checks (in the application)

- ☐ Canvas displays the portfolio/project hierarchy correctly
- ☐ Hierarchy levels render at the correct depth (L1 > L2 > L3 > L4 > L5)
- ☐ Rollup totals on parent rows match the sum of their children
- ☐ Timescale bars appear at correct dates
- ☐ Resources appear in the resource pool and can be assigned to tasks
- ☐ Picklist dropdowns in metadata forms show the expected values
- ☐ Baseline comparison displays correctly (if baseline was loaded)
- ☐ Reports run without errors (try AllTabular and AllDrilldown reports)

If the canvas does not refresh after import, click the Refresh button on the top menu. If rollup totals are incorrect, run Right-click Canvas > Recalculate All.

---

9. Post-Migration Setup Steps

After the data is confirmed correct in the application, complete the following configuration steps — these are not part of the data migration itself but are required before users can work effectively:

1. **Set the Status/Report Date** — Admin Panel > Status Date. This is the "as of" date for all progress calculations. Set it to the date of your first reporting cycle after migration.
2. **Set a Baseline** — Right-click Canvas > Set Baseline. This captures the imported schedule as the approved plan-of-record if you did not pre-populate `tbBaseline` during migration.
3. **Configure the Admin Panel** — Review product settings, calendar (working hours per day), holiday exceptions, and any product-specific options.
4. **Review CoreReport field visibility** — Using the SS Sync CoreReport tab or the in-app settings, configure which metadata fields display at each hierarchy level (L1–L5) in the Core Report view.
5. **Review and customise FOCD forms** — The Fields tab in SS Sync controls which dynamic form fields appear for each item type. Adjust to match the customer's workflow.
6. **Run resource demand calculations** — After confirming all Allocations are correct, run Recalculate All to rebuild the resource demand stores (`tbResCalcs`, `tbResCalcs2`) used by resource reports and stacked charts.
7. **Distribute the spreadsheet template** — Provide users with the downloaded `tbClient` template file for ongoing Spreadsheet Sync operations. See How to Use the Spreadsheet Sync for day-to-day sync procedures.

---

10. Cross-References

Topic	Document
Spreadsheet Sync — day-to-day use, Import All, BulkOperations tab, Setup Tab	How to Use the Spreadsheet Sync
Data hierarchy, scheduling engine, tbTimebars / tbMetaData / tbResources / tbTags JSON field reference	Data Model and Scheduling Engine Guide
Full metadata field definitions and descriptions	MetaData Fields Detail Report
Backup, restore, and data management procedures	Data Management User Guide
Risks, Issues, and Change Request fields	Risks Issues and Change Requests User Guide
Configurable features and Admin Panel	Configurable Features User Guide

Cross-reference note for the Spreadsheet Sync guide: This migration document is companion reading for the Bulk Operations import section of How to Use the Spreadsheet Sync. That guide explains how to operate the Import All function, configure the Setup Tab location, and use the BulkOperations validation metadata. Read both documents together when planning a migration.
