# NCC – ESTIMATE UPLOAD

## 20251129:0725 – CRITERIA AND OUTLINE FOR EXCEL ESTIMATE UPLOAD

NCC Estimate & SOW Design – Working Specification

1. Scope

NCC ("NEXUS Contractor Connect") needs to:

- Import detailed Xactimate estimates from Excel (XACT_RAW tab).
- Preserve a full RAW archive of each upload.
- Normalize the data into a relational model for:
  - Project hierarchy (Project → Building → Room),
  - SOW & line items,
  - Carrier vs client payer responsibility,
  - Carrier & client reconciliation flows,
  - Percent complete tracking (per line, per selection, and per project),
  - Audit history and QC across multiple estimate versions.

We focused only on the XACT_RAW worksheet in the Carrier file.

2. Excel XACT_RAW format

2.1 Columns

The headers for the XACT_RAW tab are:

1. # (line number)
2. Group Code
3. Group Description
4. Desc
5. Age
6. Condition
7. Qty
8. Item Amount
9. Reported Cost
10. Unit Cost
11. Unit
12. Coverage
13. Activity
14. Worker's Wage
15. Labor burden
16. Labor Overhead
17. Material
18. Equipment
19. Market Conditions
20. Labor Minimum
21. Sales Tax
22. RCV
23. Life
24. Depreciation Type
25. Depreciation Amount
26. Recoverable
27. ACV
28. Tax
29. Replace
30. Cat
31. Sel

32. Owner
33. Original Vendor
34. Source Name
35. Date
36. Note 1
37. ADJ_SOURCE

We will import all of these into a RAW archive table unchanged.

2.2 Hierarchy conventions

You defined the hierarchy as:

- Project name (in NCC) = geographic street address / overall job.
- Building / structure (sub-project):
  - Typically represented by Group Code (B).
- Room / area (sub-sub-project / P&L bucket):
  - Represented by Group Description (C).
  - This is the true bucket for SOW and line item costs.

Line items live under the room/area (C), and may or may not have a useful Group Code (B).

3. Data model layers

We split the design into three layers:

1. RAW layer – immutable archive of every upload.
2. MODEL layer – relational, normalized tables NCC queries and updates.
3. AUDIT/QC layer – history of changes (grouping, reconciliation, progress), and cross-version QC.

3.1 RAW layer

### 3.1.1 estimate_versions

Represents each file / version of an estimate for a project:

- id
- project_id
- source_type – e.g. xact_raw_carrier
- file_name
- stored_path
- estimate_kind – initial, carrier_supplement, client_change_order, other
- sequence_no – version order per project (0 = initial, 1 = first supplement, etc.)
- default_payer_type – carrier or client
- description – human label
- imported_by_user_id
- imported_at
- status – pending, completed, failed
- Timestamps

### 3.1.2 raw_xact_rows

Raw copy of each XACT_RAW row:

- id
- estimate_version_id
- line_no – #
- All 36 data columns from XACT_RAW:
  - group_code, group_description, desc, age, condition, qty, item_amount, reported_cost, unit_cost, unit, coverage, activity, workers_wage, labor_burden, labor_overhead, material, equipment, market_conditions, labor_minimum, sales_tax, rcv, life, depreciation_type, depreciation_amount, recoverable, acv, tax, replace, cat, sel,

owner, original_vendor, source_name, date, note_1, adj_source.

- raw_row – optional JSON snapshot
- Timestamps

RAW rows are append-only, never modified.

### 3.2 MODEL layer – Project hierarchy and SOW

#### 3.2.1 Project hierarchy

We separate building-level groups and room-level groups.

building_groups

- id
- project_id
- raw_code – from Group Code
- raw_description – optional
- display_name – editable alias (e.g. “Unit 1”, “Building A”)
- Timestamps

room_groups (primary P&L units)

- id
- project_id
- building_group_id (nullable)
- raw_name – from Group Description
- display_name – editable alias (e.g. “Unit 1 – Hallway”)
- Timestamps

Rules:

- Every SOW line must belong to a room_group.
- It may optionally be linked to a building_group (if Group Code is present).
- If estimators are sloppy (blank Group Code), the room_group still exists and can be re-assigned to a building later.

3.2.2 SOW headers & logical identity

sows

One SOW per estimate version per project:

- id
- project_id
- estimate_version_id
- source_type
- total_amount (cached sum)
- Timestamps

sow_logical_items

Logical identity of a line item across versions, even if the carrier renumbers it:

- id
- project_id
- room_group_id
- signature_hash – hash of the “stable” columns:
  - Group Description (C)
  - Desc (D)
  - Qty (G)
  - Item Amount (H)

- Unit Cost (J)
- Unit (K)
- Activity (M)
- Sales Tax (U)
- RCV (V)
- ACV (AA)
- Cat (AD)
- Sel (AE)
- Timestamps

When importing a new estimate version, we compute the signature and either:

- Reuse an existing sow_logical_items.id (same logical line), or
- Create a new one.

3.2.3 sow_items – normalized estimate lines

Each item is a normalized, query-ready copy of a RAW row, tied to a logical item and version:

- id
- sow_id
- estimate_version_id
- raw_row_id
- logical_item_id
- line_no – the # for this version
- building_group_id (nullable)
- room_group_id (required)
- Key fields (normalized from raw):
  - description (desc)
  - qty
  - unit
  - unit_cost
  - item_amount

- rcv_amount
- acv_amount
- depreciation_amount
- sales_tax_amount
- category_code (cat)
- selection_code (sel)
- activity
- material_amount
- equipment_amount
- etc., as needed.

- Payer & performance:
- payer_type – carrier or client
  - Initialized from estimate_version.default_payer_type.
- performed – bool (did we do this work?)
- eligible_for_acv_refund – bool
- acv_refund_amount – numeric (ACV * 0.8 when unperformed and eligible).

- Progress:
- percent_complete – decimal (0–1; exposes 0–100% in UI).

- Timestamps

3.3 QC: line number history

Because carriers may renumber lines across versions, even when they're logically the same:

line_number_history (or estimate_item_events)

- id
- logical_item_id
- old_estimate_version_id
- old_line_no
- new_estimate_version_id

- new_line_no
- detected_at
- Timestamps

When a new estimate version is imported:

- For each row, if we match an existing logical_item_id and the previous line_no differs, we log a history entry instead of overwriting.

This gives you a complete record of renumberings.

4. Carrier reconciliation

We need to track how the carrier responds to each proposed line.

4.1 carrier_line_decisions

Per item, per version:

- id
- estimate_version_id
- sow_item_id
- decision_status – proposed, approved, modified, denied
- carrier_qty (nullable)
- carrier_unit_cost (nullable)
- carrier_total_amount (nullable)
- notes – text
- decided_at
- decided_by – string or FK
- Timestamps

4.2 carrier_reconciliation_events

Event stream for audit:

- id
- estimate_version_id
- sow_item_id
- user_id (internal staff)
- event_type – e.g. line_added_for_carrier, line_approved, line_modified, line_denied
- payload – JSON (before/after qty, cost, totals, etc.)
- created_at

This provides a full audit trail for how each line moved through the carrier negotiation.

5. Client reconciliation & ACV / O&P logic

After job completion, we reconcile between:

- Carrier-approved amounts.
- Client change orders (client-paid).
- ACV refunds for unperformed work.

Key ideas:

- ACV (Actual Cash Value) is already per line in sow_items.acv_amount.
- The carrier distributes O&P across each line; when refunding ACV to the client for unperformed work, you remove O&P:

text

We model:

- sow_items.payer_type – who pays for this line (carrier vs client).

- sow_items.performed – whether the SOW line was actually completed.
- sow_items.eligible_for_acv_refund – can this line generate ACV refund.
- sow_items.acv_refund_amount – computed/stored as acv_amount * 0.8 when appropriate.

5.1 Optional snapshot table

client_reconciliations:

- id
- project_id
- reconciled_at
- reconciled_by_user_id
- summary_carrier_approved
- summary_client_change_orders
- summary_acv_refund
- notes
- Timestamps

Most calculations can be derived on the fly from sow_items + carrier decisions; snapshots are optional.

6. Group & structure history

We want to preserve how groups were structured over time and who changed them.

group_events

- id
- project_id
- user_id
- event_type – e.g.:

- building_group_created
- room_group_created
- room_group_renamed
- room_group_reassigned_building
- room_groups_merged

- entity_type – building_group or room_group
- entity_id
- payload – JSON (before/after)
- created_at

This gives a “timeline view” of how project hierarchy changed after import.

7. Progress / Percent complete

7.1 Storage

On each sow_item:

- percent_complete – decimal (0–1)
- Optionally, basis_amount_source (e.g. rcv, carrier_approved) if we want to be explicit.

7.2 Calculations

We use basis-weighted percentages, not simple averages:

- For any set of items (S):
  - basis_amount per item, e.g. carrier approved amount or rcv_amount.

$$
\text{percent}(S) = \frac{\sum_{i \in S} (\text{basis_amount}_i \times \text{percent_complete}_i)}
$$

$$
\sum_{i \in S} \text{basis_amount}_i
$$

]

Use this to compute:

- Project Percent Complete – all items for the project.
- Current Selection Percentage – items matching current filters.
- Sub-project (room/building) Percent Complete – items in that group only.

7.3 Update mechanisms

Two ways to update:

(a) Line-item level

PATCH /api/projects/{project}/sow-items/{sow_item}/percent-complete

- Body: { "percent": 75 }
- Updates only that item.
- Logs an event (sow_item_percent_updated).

(b) Bulk update by filter (filtered selection only)

POST /api/projects/{project}/progress/bulk-update

- Body:

json

- Applies the operation to all sow_items matching the filters.
- Logs a bulk event:
  - bulk_percent_update with filters, operation, percent, and affected_items_count.

7.4 Read API for progress

GET /api/projects/{project}/progress?{filters}

Returns:

- project_percent_complete – overall.
- selection_percent_complete – based on filters.
- Per-group data (e.g. room_groups):
  - room_group_id, name, percent_complete, tasks_count, basis_total, basis_completed.
- Line items (if needed):
  - Each item with sow_item_id, line_no, description, percent_complete, unit, qty, basis_amount, etc.

This is what powers the “NCC Project Percent Complete” UI mock (two KPIs, filters, grouped table).

8. Line matching & QC across versions

Because the carrier may renumber lines, we:

- Define logical identity via sow_logical_items:
  - Using hash of stable columns (C, D, G, H, J, K, M, U, V, AA, AD, AE) within each room group.
- On new imports:
  - If we find a matching signature_hash and room, we reuse the logical_item_id and log any line_no change in line_number_history.
  - If there’s no match, it’s a new logical line.

The QC logic can then classify:

- Renumbered lines – same signature, different line_no.
- Changed lines – same room & maybe desc, but key fields differ (qty, cost, etc.).
- New / removed lines – present in one version, not in the other.

9. Payer dimension and tags

For every estimate version and line item:

- estimate_versions.default_payer_type – predominant payer for that version.
- sow_items.payer_type – actual payer for each line (defaults from version, can be overridden).

We can also use the existing polymorphic tags system to label:

- estimate_versions, sow_items, etc. with things like:
  - carrier:approved, carrier:pending, client:upgrade, code-upgrade, etc.

This document should be a good starting point to paste into your own spec. When you're ready, we can start:

- Translating these into Laravel migrations & Eloquent models.
- Then building the XACT_RAW importer and the first NCC UIs (estimate import, group management, project percent complete).

# DISCLOSURES

## PLEASE ENTER ANY INFORMATION YOU MAY ALREADY KNOW ABOUT THE SUBJECT "PROPERTY"

*This will provide any previous knowledge to be revealed that might be considered confidential otherwise*