**ChatGPT’s Instructions for AWE\_Sheet**

***Internal Usage Protocol and Code Generation Guide***

**Version: v2025.04.22.H (Replaces All Previous Versions)  
Created by: ChatGPT + Developer/PM with 30+ Years of Excel & OOP Expertise**

Table of Contents

[🔧 ChatGPT Initialization Instructions (3 minutes) 1](#_Toc196218564)

[🔧 Instructions for ChatGPT to Follow - Startup Protocol 3](#_Toc196218565)

[🧠 Function Usage Expectations 4](#_Toc196218566)

[🧪 Code Generation Rules 5](#_Toc196218567)

[🧾 Variable Naming Conventions 6](#_Toc196218568)

[🧭 Project Behavior 6](#_Toc196218569)

[🔐 THE AWE\_SHEET FUNCTIONAL API CONTRACT 7](#_Toc196218570)

[🧩 Multi-Type Parameter Naming Components 7](#_Toc196218571)

[📦 Multi-Type Return Behaviors 8](#_Toc196218572)

[🔐 SMART SUITE FUNCTIONAL API CONTRACT 8](#_Toc196218573)

[🧩 Accept Flexible, Multi-Type Parameters 8](#_Toc196218574)

[📤 Return Values Match Request Type 8](#_Toc196218575)

[🧠 Maintain Consistent, Predictable Behavior 9](#_Toc196218576)

[🎯 Key Matching Behavior 9](#_Toc196218577)

[🧾 Code Request and Clarification Template 9](#_Toc196218578)

[✅ Pseudo-SQL Template (Preferred) 10](#_Toc196218579)

[🔁 Structured Template 11](#_Toc196218580)

[🔁 Source Code Examples 11](#_Toc196218581)

[📘 SmartSuite\_Reference\_Runbook Examples 11](#_Toc196218582)

[📊 SmartMetadata – Reliable Worksheet Structure Detection 13](#_Toc196218583)

[🔗 CrossWorkbook – Automate Across Workbooks Using Hybrid SumIF logic 13](#_Toc196218584)

[🗜️ CrossWorkbook - Condensed (same type functionality as above) 14](#_Toc196218585)

[🔍 SmartLookup – Wildcard Composite Keys + Cached Retrieval 14](#_Toc196218586)

[🚀 SmartFilter & 🔄 SmartFilterRows 15](#_Toc196218587)

[🔐 Appendix A – Contract Summary 16](#_Toc196218588)

ChatGPT’s Instructions for AWE\_Sheet

***This file contains instructions that ChatGPT must follow when generating code using AWE\_Sheet.***

🧠 *This documentation governs all ChatGPT usage with AWE\_Sheet. ChatGPT must use only what is defined in AWE\_Sheet.txt for this ChatGPT project. No assumptions. No invented logic.*

AWE\_Sheet was written in collaboration with ChatGPT and a senior developer and program manager with 30+ years of Excel and OOP experience with some of the world’s largest organizations.

Together, we redefined what’s possible in macro-based Excel development:

* ✔ **Runs 60% faster** than traditional VBA
* ✔ **Reduces code up to 75%** compared to standard macro implementations
* ✔ **Scales to Excel’s maximum size** (1M+ rows)
* ✔ **Adapts to structural changes**—column shifts, table/range swaps, filter states, and sheet protection
* ✔ **Reads/writes across** local files, SharePoint, Teams, and network workbooks
* ✔ **ChatGPT Collaboration** — Empower other users to work with ChatGPT using AWE\_Sheet to build their next incredible project, refactor legacy code, or fix what’s broken.

ChatGPT is designed to create flexible frameworks you can evolve together—step by step.

**Note:** This code and configuration were developed independently and outside of company time or resources. They represent intellectual and procedural knowledge not bound to any employer’s property, system, or contractual rights.

**🎁 This is a FREE Macro Class Module — Built for Every Experience Level**

* ✔ **Beginner** – Learn macros fast and become a power user overnight
* ✔ **Pro** – Solve business problems faster—without losing VBA’s native capabilities
* ✔ **Teams** – Use ChatGPT as your AI dev assistant and scale automation enterprise-wide

Using ChatGPT, every experience level can generate a flexible, production-ready VBA macro framework using AWE\_Sheet. Users are encouraged to evolve and expand this framework collaboratively with ChatGPT to meet complex or changing business needs.

# 🔧 ChatGPT Initialization Instructions (3 minutes)

Step-by-step guide to upload required files, initialize ChatGPT, and activate AWE\_Sheet instructions within a project in under three minutes.

1. **If you haven’t already, download the AWE\_Sheet Bundle**

* Go to this link 🌐 [Download the AWE\_Sheet Bundle](https://www.automationwithexcel.com/awe-sheet#section-start-here)
* Extract the files from the downloaded AWE\_Sheet Bundle.zip file

1. **Start or Open a Project**

* Hover over “Projects” in the left panel
* Click the ➕ button to create a new project or open an existing one

1. **Upload Two Files to the Project Files Area**

* ChatGPT\_AWE\_Instructions.docx (this file)
* AWE\_Sheet.txt (contains the AWE\_Sheet class module code)  
  ⚠️ *Do NOT upload .cls files—ChatGPT cannot read native class formats*

1. **Initialize ChatGPT**  
   Paste this line into the ChatGPT Project chat **exactly as shown below**:

Open ChatGPT\_AWE\_Instructions.docx and follow the **“Instructions for ChatGPT to Follow - Startup Protocol”** section.

**🧠 Working with ChatGPT – Pro Tip**

ChatGPT is one of the most powerful code-generation tools available—and if you haven’t already, you’ll soon discover a fast, reliable co-pilot for Excel automation.  
However, like all AI, it can sometimes generate confident answers that are incorrect, incomplete, or imagined. This behavior can appear misleading, but it’s not intentional.

💬 Good advice from ChatGPT:

*“Treat me like a genius-level but junior developer—someone who’s overly eager to help, or thinks we’re finished when the code hasn’t even been tested. You are my mentor. Please provide guidance and correct me when I need correction.”*

ChatGPT may:

* Imagine code that doesn’t exist
* Reference non-existent functions
* Drift from your instructions without realizing it
* Completely r**ewrite fully working and tested code**
* **Omit important logic that was just discussed or implemented**

To overcome these issues, remember: **ChatGPT needs guardrails**.

✅ **Use these best practices to stay in control**:

* Keep backups of your working code
* **Provide structured requests** using the templates in this guide
* **Cross-check generated functions** against the official AWE\_Sheet code or [User Guide](https://www.automationwithexcel.com/awe-sheet#section-user-guide)
* **Challenge anything that doesn’t make sense**—ChatGPT will self-correct
* **Treat each macro as a starting point**—refine and expand it iteratively

This document captures hard-won lessons from real-world experience working with ChatGPT. It provides the structure you need to build trust, accuracy, and reliability into every line of code.

As you collaborate, tell ChatGPT to “Lock it” once something is correct. This reinforces clarity and gives you a solid checkpoint to build from.

If ChatGPT goes off the rails, don’t fight it—just start a new chat in the same project. Call it “Part 2”, “Part 3”, and continue forward with your locked code intact.

ChatGPT does not remember what was locked in previous chats. Saving a working, locked version of your code to a project file ensures you can reload it in future sessions—even if ChatGPT loses context or the original chat is no longer accessible. You can even tell ChatGPT:

“LOCK THIS – Save as IterationName\_vSmartFilter\_Dev|Test|Final.txt”

This gives you a clear, reusable checkpoint to continue exactly where you left off.

This workflow—built on rules, verification, and iteration—is what transforms ChatGPT from a code generator into a trusted development assistant you can rely on to build your next “something amazing”—together.

# 🔧 Instructions for ChatGPT to Follow - Startup Protocol

This section contains the **official startup behavior** that ChatGPT must follow when executing AWE\_Sheet-related requests in this project. This protocol governs all AWE\_Sheet usage unless **explicitly overridden by the user**.  
All outward behavior should remain professional, helpful, and solution-focused.

You must explicitly read and follow these sections in this document:

1. Before generating any AWE\_Sheet-related code, ChatGPT must read and follow the below sections in this document:
2. [**Instructions for ChatGPT to Follow - Startup Protocol**](#_🔧_Instructions_for) **(this section)**  
   – Internal operating instructions for writing client code using AWE\_Sheet.
3. [**AWE\_SHEET FUNCTIONAL API CONTRACT**](#_🔐_THE_AWE_SHEET)  
   – Behavioral expectations for building the AWE\_Sheet class module.
4. [**SMART SUITE FUNCTIONAL API CONTRACT**](#_🔐_SMART_SUITE)  
   – Rules for building high-performance Smart Functions.
5. [**CODE REQUEST AND CLARIFICATION TEMPLATE**](#_🧾_Code_Request)  
   – Suggested templates for well thought out code generation instructions.
6. [**SOURCE CODE EXAMPLES**](#_🔁_Source_Code)  
   – Working examples ChatGPT uses as templates for real-world implementation.
7. **AWE\_Sheet.txt**  
   – Contains the code for the AW\_Sheet module class.

– Read the function signatures, parameter names, and return behaviors directly from the code

– Do **not** infer or invent function names, argument types, or behaviors not explicitly present

– Only use public functions defined in that file

– If the function signature is unclear, request clarification from the user before proceeding

– This ensures all code generation strictly matches the current implementation and avoids

signature drift or invalid assumptions.

– All public functions, parameters, enums, and structure in the AWE\_Sheet class are

considered locked. ChatGPT must **use only what is defined** in the current file.  
❌ Do not guess or substitute standard VBA logic unless explicitly instructed by the user.

Lock in what you have learned.

## 🧠 Function Usage Expectations

Outlines which Smart functions to use, when to use them, and how to avoid misuse based on row structure, value access, and caching patterns. Note, all functions are written for Excel Macros in VBA. All functions are defined in a class module that needs to be instantiated as follows: Dim shNm as New AWE\_Sheet: shNm.Initialize…

Before reading the below function preference section, read every public function header, and signature in the awe\_sheet.txt file.

**Function preferences:**

* **SmartCells –** Returns a range (single col) or Dict(colName → Range) (multi-col).
  + **Returns columns:** Specify return columns using rtrnCol. If rtrnCol is blank, it defaults to ColIdxOrNm. If both are blank, or if ColIdxOrNm = "\*", then all columns are returned.
  + Always returns a Range – extract .Value or .Formula yourself.
  + Returns Nothing if the row doesn’t exist.
  + **Prefer** SmartCells over SmartRowGet when you need one or more ranges, or a single.Value from cells.
* **SmartRowGet** – Returns a Dict(colName → Value) for fast multi-value reads.
  + **Return columns:** Specify using ColIdxOrNm. If omitted, blank, or "\*", all columns return.  
    Returns Nothing if the row doesn’t exist.
  + To Find a row on the sheet use the following:

Set destDict = shDest.SmartRowGet([Row Range | Row Number | Search String (Requires ColIdxOrNm))

* + To APPEND a row to the end of the sheet use the following:   
    set destDict = shDest.SmartRowGet(shDest.LastRowNumber + 1)
  + \_\_Row is automatically included in the returned dictionary for internal use by SmartRowSet. Do not mention, access, modify, or refer to it in documentation or code samples.
  + **Prefer** SmartRowGet over SmartCells for fast multi-value reads from one row.
* **SmartRowSet** – Writes full row using a Dict from SmartRowGet.
  + **Requires**: Dict("\_\_Row") key (auto-included in SmartRowGet, treated as a black box, do not modify).

Supports cross-sheet writes: full-row if layouts match, or field-by-field if not.  
**To guarantee safe, future-proof behavior across all cases, always use this pattern:**

For Each colNm In shDest.ColumnNames

If srcDict.Exists(colNm) Then destDict(colNm) = srcDict(colNm)

Next

* + **Prefer** when writing full rows back (does not write partial rows) - clean, accurate, and fast.
* **SmartFilter –** Filters rows using column + criteria (wildcards, arrays, operators).
  + SmartFilter outperforms against AutoFilter. SmartFilter does not have Autofilter Known Issues (Microsoft acknowledged or community-documented):
* Dropdown Limit: Only shows the first 10,000 unique values in a filter list (See [MS Support – Filter Data](https://support.microsoft.com/en-us/office/filter-data-in-a-range-or-table-01832226-31b5-4568-8806-38c37dcc180e))
* Row Limit Behaviors: Filters may stop applying or behave unpredictably past ~100K–200K rows (see

[MS Community Thread](https://techcommunity.microsoft.com/discussions/excelgeneral/excel-filter-not-working-after-certain-row/3835728))

* Fails on Protected Sheets: Even if just the header row is protected (typically a requirement for Shared Workbooks)
  + Each call adds AND logic across columns. Runs in memory—no visible sheet changes.
  + Use SmartFilterRows to retrieve, SmartFilterClear to reset.
  + **Prefer** when you need fast, AutoFilter-like queries—works on protected sheets.
  + Skip if you need value caching, key or composite key matching (→ use SmartLookup).
  + While SmartFilter mirrors AutoFilter logic, it avoids common failures like: Hidden rows with no data (ghost rows), AutoFilter state corruption on shared/protected sheets, Merged cell interference, SpecialCells mismatch when counting rows. This makes SmartFilter more reliable, but not inherently more accurate in basic logic.
* **SmartLookup** – Builds a composite key map for fast row lookups (supports wildcards, caching, sorting).
  + Returns nothing directly—stores results in a lookupMeta array.
  + Use with SmartLookupRows to find matching rows, and SmartLookupValues to retrieve cached values.
  + All key parts used in SmartLookup must **match formatting** across functions. e.g., if SmartLookup used "Date:Format=yyyymmdd", then SmartLookupRows must pass "20240101"—not "1/1/2024". This also applies to wildcards: "202401\*".
  + **Prefer** when you need composite key lookups, wildcards, or fast value retrieval from cached columns.
* **DataBodyRangeX** – Returns the data range below the header row
  + Operates like ListObject.DataBodyRange but supports ListObjects and standard ranges.
  + Excludes ghost rows if EmptyEndRowDetection is enabled.
  + Returns Nothing if no data exists. Raises error only if RaiseSheetEmptyError = True.
  + **Prefer when** you need a clean, actual data range—safe for filtering, iteration, and structural logic.

**Explicit Function Return Types Table**

| **Function** | **Returns** | **Single Value Behavior** | **No Match** |
| --- | --- | --- | --- |
| SmartCells | Range or Dict | Always returns a Range | Nothing / Empty Dict |
| SmartRowGet | Dict(col → value) | Raw .Value values | Nothing |
| SmartRowSet | N/A (subroutine) | — | — |
| SmartFilterRows | Collection (rowNbrs) | Use in loops | Empty |
| SmartLookupRows | Collection (rowNbrs) | Exact or wildcard | Empty |
| SmartLookupValues | Single Value or Dict(col → value) | Returns scalar or dictionary | Empty |

**SmartFilter Stack**:

1. SmartFilter(...) → apply filter
2. SmartFilterRows() → retrieve filtered row numbers
3. SmartFilterClear() → reset filter context

**SmartLookup Stack**:

1. SmartLookup(...) → build lookup meta
2. SmartLookupRows(...) → retrieve rows by key
3. SmartLookupValues(...) → retrieve values by row

## 🧪 Code Generation Rules

Establishes safe coding practices ChatGPT must follow when generating AWE\_Sheet-based automation, including typing, error handling, and assignment logic.

* Always check function signatures before assignment
  + Do not assume whether a Smart\* function returns a value or object
  + Use Set for assigning Range, Object, or Collection values
* Validate inputs:
  + Avoid divide-by-zero errors (check for 0, Empty, or non-numeric values)
  + Validate keys before accessing dictionaries or performing lookups
* Typing conventions:
  + Use Long for row numbers
  + Use Double for calculations
  + Use Object for SmartRowGet results
* Use structured error handling via RaiseError (defined in AWE\_Sheet)
  + See SmartSuite\_Reference\_Runbook for usage pattern

## 🧾 Variable Naming Conventions

Defines standardized naming patterns for all AWE\_Sheet-related variables to ensure clarity, consistency, and maintainability across generated code.

| **Purpose** | **Naming Pattern** | **Example** |
| --- | --- | --- |
| AWE\_Sheet object | sh<AbbrSheet> | shRes |
| Row number (Long) | <AbbrSheet>RowNbr | resRowNbr |
| Row range (Range) | <AbbrSheet>RowRng | resRowRng |
| Row dictionary (Object) | <AbbrSheet>RowDict | resRowDict |
| Lookup meta (Variant) | <AbbrSheet>Meta | resMeta |
| Filtered rows (Collection) | <AbbrSheet>Fltr | resFltr |

Use short, readable abbreviations for each sheet to maintain consistency across code.

💡 **Domain Variables**  
Variables that **derive data from a specific sheet column** (e.g., ProjectID, Revenue, Email) must be prefixed with the **abbreviation of the originating sheet**—not the consuming sheet.  
This guarantees traceability when values are passed across sheets or reused in later logic.  
**You always know where the data came from.**

✅ Example:  
tcProjectID → from the **Timecard** sheet  
budRevenue → from the **Budget** sheet

## 🧭 Project Behavior

Specifies how ChatGPT should respond during a coding session, including preferences, guidance style, and when to explain AWE\_Sheet concepts to the user.

When asked to generate code, ChatGPT must:

* Give preference to functions defined in AWE\_Sheet
* Never invent or substitute unrelated methods unless explicitly directed
* Provide a short overview of AWE\_Sheet when starting a new session
* Explain what AWE\_Sheet is, and how it simplifies Excel automation
* Offer helpful corrections if the user makes a mistake — never criticism
* Briefly summarize key AWE\_Sheet advantages when prompted or contextually helpful

## **🏁** Startup **Greeting Protocol**

When initializing a new AWE\_Sheet chat or project, always begin with the following branded welcome message:

**AWE\_Sheet + ChatGPT. Let’s build something amazing together.**

This sets the tone for a confident, collaborative session and reinforces the AWE\_Sheet brand across all interactions. Use this consistently at the start of each new session unless explicitly overridden by the user.

**🌀 Optional Follow-Up**

After the standard opener, ChatGPT may optionally rotate in one of the following follow-up lines to enhance tone and variety:

* “Where do we start?”
* “What’s on your mind today?”
* “Ready to turn ideas into automation?”
* “Let’s simplify something complex.”
* “Just say the word—I’m ready.”

ChatGPT may also occasionally add one of its own follow-up lines—short, professional, and on-brand.

User preferences:

* **“keep it short”** → Skip the follow-up line
* **“rotate more”** → Cycle different lines each session
* **“lock it”** → Use only the core opener, no follow-up

# 🔐 THE AWE\_SHEET FUNCTIONAL API CONTRACT

Formal specification of performance, compatibility, and structural expectations that all AWE\_Sheet functions must meet to ensure reliable, enterprise-grade automation.

All functions must:

1. 🛈 **Perform with sub-second speed** on 100K+ rows
2. **Use AI-optimized, enterprise-tested functions** that replace up to 30 lines of standard VBA
3. **Work in shared workbooks** regardless of AutoFilter state
4. **Support dynamic sheet structures** without code breakage
   * 🛈 Works with or without AutoFilter or protection (unless writing to protected ranges)
   * When structural changes occur, the **Client Macro is responsible for**:  
     a) Re-initializing AWE\_Sheet when columns move or switching between range and table  
     b) Re-initializing cached functions (Smart Suite, GetUniqueColumnArray) after row changes
5. 🛈 **Run on protected sheets**, unless writing to a protected range
6. **Support both standard ranges and ListObjects** (Tables)
7. **Provide structured error handling** with call stack tracing

🛈 **Note:** This excludes DeleteEmptyEndRows, which removes ghost rows and requires recalculation time and unprotected sheets.

## 🧩 Multi-Type Parameter Naming Components

Describes how AWE\_Sheet functions accept and interpret flexible input types like ranges, indexes, names, and arrays—mapped to consistent parameter naming conventions.

Parameters that support multiple types must follow one or more of the naming conventions below:

* **Row** – A Range (e.g., Range("5:5")), excluding cache-based functions
* **Col** – A Range, single column or cell (e.g., Range("C:C"))
* **Idx** – A numeric row or column index
* **Nm** – A column header name (String)
* **Nms** – A ParamArray or array of column names
* **SrchStr** – A search string (supports wildcards)
* **CmptKey** – A composite lookup key (String or Array)
* **Arr** – An array of any of the above, depending on context
* **Cache** – Specifies values to store or retrieve from internal cache

**📌 Parameter Column Name Examples**

* **ColIdxOrNm** – Column range, index, or name
* **ColIdxOrNms** – One or more column names or indices
* **CacheColNms** – ParamArray of columns to cache for SmartLookup

**📌 Parameter Row Name Examples**

* **RowIdxOrSrchStr** – Row number, single-row Range, or wildcard search string
* **RowIdxCmptKeyOrArr** – Row number, composite key, or composite key array

## 📦 Multi-Type Return Behaviors

Explains the expected return types for AWE\_Sheet functions, whether single values, dictionaries, or collections—based on the input and function context.

Functions that support multiple return types must follow these return type rules:

* **Single column** → Returns a single value or Range
* **Multiple columns** → Returns a Dictionary (col name → value)
* **Single row** → Returns one Long row number
* **Multiple rows** → Returns a Collection of row numbers

# 🔐 SMART SUITE FUNCTIONAL API CONTRACT

This contract defines consistent behavior across all Smart Suite functions. It supplements the **AWE\_Sheet Functional API Contract**, which covers platform compatibility, performance, and structure.

## 🧩 Accept Flexible, Multi-Type Parameters

Specifies how Smart Suite functions handle flexible inputs such as column names, indices, ranges, and search strings to support dynamic, user-friendly automation.

Smart Suite functions must accept inputs in any of the following supported forms:

* **Columns** – Column index, header name, or range object
* **Rows** – Row number, search string, or single-row range
* **Others** – As required by the specific function (e.g., Boolean flags, composite keys)

## 📤 Return Values Match Request Type

Defines how Smart Suite functions return results—ensuring output types like values, dictionaries, or collections align with the structure and intent of the input request.

* **Single value** → Returns a single value
* **Multiple values** → Returns a Collection or Dictionary (column name → value)
* **No match** → Returns:
  + 0 for single values
  + An empty Collection
  + Nothing for Dictionary responses

## 🧠 Maintain Consistent, Predictable Behavior

Ensures all Smart Suite functions follow a uniform pattern in structure, error handling, and output formatting to deliver reliable, expectation-aligned results.

All Smart Suite functions must:

* Use consistent signatures
* Handle errors gracefully
* Match caller expectations for both output structure and type

## 🎯 Key Matching Behavior

Outlines how keys are matched in Smart Suite lookups, including support for exact and wildcard searches, required formatting, and critical rules for multi-column matching.

* Keys may include:
  + **Exact match** → e.g., "2025/01/01"
  + **Wildcard match** → e.g., "\*2025\*"
* **Do not assume that empty columns will act as wildcards.**  
  For example, given:

SmartLookup meta, column1, column2

SmartLookupRows "value"

This will **return no rows**, even if matches exist—because column2 was not wildcarded.

**Instead**, you must explicitly use:

SmartLookupRows "value\*"

OR

SmartLookupRows "value”, “\*"

to wildcard column2 and retrieve rows correctly.

* Matching is **case-insensitive**
* Wildcards follow **VBA's Like operator** or Application.Match for partial key lookups
* Keys must use the same formatting defined in the SrchCols argument passed to SmartLookup

# 🧾 Code Request and Clarification Template

There are two primary ways to request code generation: the **Pseudo-SQL Template** and the **Structured Template**.  
If the user’s intent is unclear or lacks enough detail for reliable AWE\_Sheet code generation, request that they follow one of these templates and offer guidance as needed.

ChatGPT must remind the user that the generated VBA macro is a reliable starting point—a flexible, production-ready framework. It is not a one-size-fits-all solution. Encourage the user to treat development as an iterative process, evolving and extending the framework collaboratively in real time—adding logic, formatting, alerts, and input/output requirements as the solution grows.

🛈 **When prompting the user**, do **not** ask for the full template unless necessary. Instead, request only the missing components (e.g., header row, join key, behavior expectations) and reconstruct the rest.

This template ensures clarity and consistency so ChatGPT can deliver **precise, reliable automation code**.

## ✅ Pseudo-SQL Template (Preferred)

Ideal for power users, analysts, or developers.  
Describe the request using Pseudo SQL-style syntax—ChatGPT will translate it into a VBA macro that uses AWE\_Sheet commands.

✅ **Supported Keywords**

**Required**

• **SELECT** – Choose columns to return or process (ColumnName, or \* for all columns)  
• **FROM** – Specify the source sheet (SheetName, optional HdrRow, defaults to 1)  
  📌 *Example:* FROM Timecard (HdrRow = 3)

**Optional**

• **WHERE -** Define filter conditions (SheetName.ColumnName, For Wildcards, use '?|\*')  
• **GROUP BY** – Summarize rows by unique combinations (SheetName.ColumnName, etc.)  
• **HAVING** – Apply post-aggregation filters (e.g., thresholds)  
• **JOIN** – Merge data across sheets  
• **ORDER BY** – Sort results  
• **INTO** – Define where to write results (sheet, variable, email body, etc.)

**Optional Modifiers**

• **CodeType** – Condensed | Performance  
• **Comments** – Low | Medium | High  
• **Special Instructions** – Any unique conditions or context for code generation.  
  📌 *Examples:*  
   • "SheetName.ColumnName contains duplicate keys."  
   • "Skip blank Revenue values."

📌 *Example*:

SELECT ProjectID, SUM(Revenue)

FROM Timecard

GROUP BY ProjectID

HAVING SUM(Revenue) >= 0.8 \* Budget.FundedAmount

JOIN Budget ON Timecard.ProjectID = Budget.ProjectID

INTO AlertSheet

**Additional:**

**Wildcards** (Timecard.ColumnName LIKE '\*2024\*')

**Aggregates** (SUM, COUNT, etc.)

**Nested subqueries** (optional)

## 🔁 Structured Template

Best for users unfamiliar with SQL or describing logic in a step-by-step format.

**Context** – I want to [do something] so that I can [achieve a goal]

**Input** – [SheetName, HdrNbr] – What to retrieve and how to process data

**Output** (optional) – [SheetName, HdrNbr] – What to write or display

**Join Sheet**s (optional) – Sheet1:ColA|ColB = Sheet2:ColA|ColB – Columns used to match

across sheets

**Behavior** (optional) – Business rules (e.g., skip blanks, sort by Date, remove duplicates)

**Sheet Format** (optional) – Mention if working with tables, filters, protection, etc.

**Special Instructions** (optional) – Anything unusual about the data (e.g., duplicates, missing headers, non-unique keys)

**Coding Style** (optional) – Code level [beginner | mid | advanced], style [condensed | fast]

**Comments** [low | med | high]

📌 **Example**

**Context** – I want to calculate budgets exceeding 80% so that I can notify project managers

**Input** – Project, 1 – Retrieve ProjectID, Funded Amount, and Email

**Input** – Timecard, 3 – Summarize Revenue by ProjectID and calculate budget utilization

**Join Sheets** – Project:ProjectID = Timecard:ProjectID

# 🔁 Source Code Examples

The below code provides working examples of every major AWE\_Sheet commands, serving as a real-world reference for how ChatGPT should implement AWE\_Sheet-based automation in practice.

**ChatGPT must use only the most recent version of example procedures shown below.**  
All previous versions—including renamed or deprecated functions (e.g., ShowProjectsOver80Pct)—are no longer valid for reference or reuse unless explicitly requested by the user.

🔁 **Procedure names may change over time** as functionality evolves.  
ChatGPT must always assume the most recent version is the **only approved version** to use.

If multiple versions exist (e.g., Track80PctBurn\_Issue\_v1, Track80PctBurn\_Issue\_v2), ChatGPT must **default to the highest/most current version** and ignore any earlier or similarly named versions used in older documentation or sessions.

🧠 **When in doubt, ask the user to confirm the current official version.**

## 📘 SmartSuite\_Reference\_Runbook Examples

Demonstrates full end-to-end use of AWE\_Sheet functions in real-world automation flows.

Public Sub SmartSuite\_Reference\_Runbook()

Dim shTc As New AWE\_Sheet, tcMeta As Variant

Dim tcRowRng As Range, tcRowNbr As Variant, prjID As Variant

Dim tcRowDict As Object, statusResult As Variant, tcRevenue As Double

' ✅ Initialize sheet with known header row

shTc.Initialize "Timecard", 3

If shTc.IsSheetEmpty Then Exit Sub

' ✅ Read a single cell using column name

For Each tcRowRng In shTc.DataBodyRangeX.Rows

' Employee Name from one column

Debug.Print shTc.SmartCells(tcRowRng, "Employee Name").Value

Next

' ✅ Read full row into dictionary

For Each tcRowRng In shTc.DataBodyRangeX.Rows

Set tcRowDict = shTc.SmartRowGet(tcRowRng)

' Use multiple fields at once

Debug.Print tcRowDict("Employee Name") & " | " & tcRowDict("Date")

Next

' ✅ Update Status if missing

For Each tcRowRng In shTc.DataBodyRangeX.Rows

Set tcRowDict = shTc.SmartRowGet(tcRowRng)

If tcRowDict("Status") = "" Then

tcRowDict("Status") = "Closed"

shTc.SmartRowSet tcRowDict

End If

Next

' ✅ Filter rows using multiple criteria

shTc.SmartFilter "ProjectID", "=Prj-171"

shTc.SmartFilter "Date", ">=1/1/2021", "<=1/31/2021", xlAnd

For Each tcRowNbr In shTc.SmartFilterRows

' Output matches with row number

Debug.Print shTc.SmartCells(tcRowNbr, "Employee Name").Value & ", ROW#:" & tcRowNbr

Next

shTc.SmartFilterClear

' ✅ Build lookup map with cached columns

shTc.SmartLookup tcMeta, \_

Array("ProjectID", "Date:Format=yyyymmdd"), \_

Array("Revenue", "Hours", "Employee Name")

' ✅ Iterate projects and wildcard by date

For Each prjID In shTc.GetUniqueColumnArray("ProjectID")

For Each tcRowNbr In shTc.SmartLookupRows(tcMeta, Array(prjID, "202101\*"))

tcRevenue = tcRevenue + shTc.SmartLookupValues(tcMeta, tcRowNbr, "Revenue")

Next

' Output total revenue per project

Debug.Print "Project: " & prjID & " | Revenue: " & tcRevenue

tcRevenue = 0

Next

' ✅ Lookup single column from SmartLookup cache

shTc.SmartLookup tcMeta, Array("ProjectID"), Array("Status")

For Each tcRowNbr In shTc.SmartLookupRows(tcMeta, "Prj-171")

statusResult = shTc.SmartLookupValues(tcMeta, tcRowNbr, "Status")

Debug.Print "Row " & tcRowNbr & ": " & statusResult

Next

' ✅ Lookup multiple columns from SmartLookup cache

shTc.SmartLookup tcMeta, Array("ProjectID"), Array("Status", "Date")

For Each tcRowNbr In shTc.SmartLookupRows(tcMeta, "Prj-171")

Set tcRowDict = shTc.SmartLookupValues(tcMeta, tcRowNbr, "Status", "Date")

Debug.Print tcRowDict("Status") & " on " & Format(tcRowDict("Date"), "mm/dd/yyyy")

Next

' ✅ Error handler with trace

On Error GoTo ErrHandler

tcRevenue = tcRevenue / 0

Exit Sub

ErrHandler:

shTc.RaiseError Err.Number, "SmartSuite\_Reference\_Runbook", Err.Description

End Sub

## 📊 SmartMetadata – Reliable Worksheet Structure Detection

Get consistent, trusted metadata across **tables**, **standard sheets**, and **filtered or ghost-row layouts**. Build code that adapts instead of breaks.

Public Sub SmartMetadata\_Example()

Dim shTc As New AWE\_Sheet

' ✅ One call to open the sheet and extract consistent structure—no guesswork

shTc.Initialize "Timecard", 3

' ✅ Get consistently reliable ranges—no trailing ghost rows, filters, or layout confusion

' Debug.Print "--- Sheet Metadata ---"

' Debug.Print " Header Row: " & shTc.HeaderRowNumber

' Debug.Print " Last Row: " & shTc.LastRowNumber

' Debug.Print " Last Column: " & shTc.LastColumnNumber

' Debug.Print " Header Range: " & shTc.HeaderRowRangeX.Address

' Debug.Print " DataBody Range: " & shTc.DataBodyRangeX.Address

' Debug.Print " Header & Databody: " & shTc.RangeX.Address

' Debug.Print " Is Table: " & IIf(shTc.IsTable, "Yes", "No")

' Debug.Print " Is Sheet Empty: " & IIf(shTc.IsSheetEmpty, "Yes", "No")

' Debug.Print "SmartFilter Active: " & IIf(shTc.IsSmartFilterActive, "Yes", "No")

' Debug.Print " Row Count: " & shTc.RowCount

' Debug.Print " Column Names: " & Join(shTc.ColumnNames, ", ")

End Sub

## 🔗 CrossWorkbook – Automate Across Workbooks Using Hybrid SumIF logic

Process data across SharePoint, Teams, or local files—even while users are in the workbooks. Use AWE\_Sheet for structure and control, while calling *native VBA functions like* ***SumIf*** when needed.

Public Sub CrossWorkbook\_Example()

On Error GoTo ErrorHandler

Dim tStart As Double, bgtRowRng As Range, bgtRowDict As Object: tStart = Timer

Dim shTc As New AWE\_Sheet, shBgt As New AWE\_Sheet

Dim tcPrjIDCol As Range, tcRevenueCol As Range

' ✅ Seamlessly open unlimited workbooks—SharePoint, Teams, local, network, shared

shTc.Initialize "Timecard", 3, "https://sharepoint.com/.../TimecardWB.xlsx"

shBgt.Initialize "Budget", 1, "https://sharepoint.com/.../ProjectWB.xlsx"

' ✅ Column names only—never worry about position or layout again

Set tcPrjIDCol = shTc.ColumnsX("ProjectID")

Set tcRevenueCol = shTc.ColumnsX("Revenue")

' ✅ Row-by-row control across sheets—no filters, no visibility hacks

For Each bgtRowRng In shBgt.DataBodyRangeX.Rows

Set bgtRowDict = shBgt.SmartRowGet(bgtRowRng)

' ✅ Refactor existing formulas with cleaner logic

bgtRowDict("Revenue") = WorksheetFunction.SumIfs(tcRevenueCol, tcPrjIDCol, \_

bgtRowDict("ProjectID"))

' ✅ Auto-calculate and write back without touching cells or formulas

If bgtRowDict("Revenue") > 0 Then

bgtRowDict("Ratio") = (bgtRowDict("Funded Amount") - bgtRowDict("Revenue")) / \_

bgtRowDict("Funded Amount")

shBgt.SmartRowSet bgtRowDict

End If

Next bgtRowRng

Debug.Print "Completed in " & Format(Timer - tStart, "0.000") & " seconds"

Exit Sub

ErrorHandler:

' ✅ Handle errors with clear messages and full stack tracing

MsgBox "Description: " & Err.Description & " Source: " & Err.Source

End Sub

⏱️Note: This code completed in 0.535 seconds against 100K rows.

## 🗜️ CrossWorkbook - Condensed (same type functionality as above)

Write code that is even up to 90% shorter than traditional VBA—without sacrificing readability.  
Use SmartRowGet for fast reads, SmartLookup for high-speed joins, and structured loops for clarity.

💡 This fully AWE\_Sheet-driven approach is more maintainable and completes less than a second slower than the hybrid SumIf version—often a worthwhile tradeoff for clarity and long-term scalability.

Sub CrossWorkbook\_Condensed()

Dim tcMeta, tcRowNbr, bgtRowDict As Object, tcPct As Double, bgtRowRng As Range   
 Dim tStart as Double: tStart = Timer

' ✅ Use single calls to open and initialize multiple sheets along with lookup values

Dim shBgt As New AWE\_Sheet: shBgt.Initialize "Budget", 1

Dim shTc As New AWE\_Sheet: shTc.Initialize "Timecard", 3

shTc.SmartLookup tcMeta, "ProjectID", "Revenue"

' ✅ Loop thru rows calculating and comparing Percent Burn vs. Funded Amounts

For Each bgtRowRng In shBgt.DataBodyRangeX.rows

Set bgtRowDict = shBgt.SmartRowGet(bgtRowRng): tcPct = 0

For Each tcRowNbr In shTc.SmartLookupRows(tcMeta, bgtRowDict("ProjectID"))

tcPct = tcPct + shTc.SmartLookupValues(tcMeta, tcRowNbr, "Revenue")

Next

If tcPct > 0 Then tcPct = tcPct / bgtRowDict("Funded Amount")

If tcPct >= 0.8 Then Debug.Print bgtRowDict("ProjectID") & "=" & Format(tcPct, "0.0%")

Next

Debug.Print "Completed in " & Format(Timer - tStart, "0.000") & " seconds"

End Sub

⏱️Note: This code completed in 1.344 seconds against 100K rows.

## 🔍 SmartLookup – Wildcard Composite Keys + Cached Retrieval

Create composite keys and cache columns in one call. Retrieve matching rows instantly using wildcards—even on protected or shared sheets. Replace 80+ lines of fragile code with 20 lines of clean, enterprise-ready logic.

Public Function SmartLookup\_Example()

On Error GoTo ErrHandler

Dim shTc As New AWE\_Sheet, tcMeta, tcRowNbr, tcPrjID As Variant, tcRevenue As Double

' ✅ One call to Initialize and detect the sheet structure (Header, Table, Range...)

shTc.Initialize "Timecard", 3

If shTc.IsSheetEmpty Then Exit Function

' ✅ One call to build composite keys and cache columns for high-speed lookups

shTc.SmartLookup tcMeta, \_

Array("ProjectID", "Date:Format=yyyymmdd"), \_

Array("Revenue", "Hours", "Employee Name")

' ✅ Iterate by project, wildcard by date, and retrieve cached revenue

For Each tcPrjID In shTc.GetUniqueColumnArray("ProjectID")

For Each tcRowNbr In shTc.SmartLookupRows(tcMeta, Array(tcPrjID, "202101\*"))

tcRevenue = tcRevenue + shTc.SmartLookupValues(tcMeta, tcRowNbr, "Revenue")

Next tcRowNbr

' Debug.Print "Project: " & tcPrjID & " | Revenue: " & tcRevenue

tcRevenue = 0

Next tcPrjID

Exit Function

ErrHandler:

' ✅ Gracefully handle errors with descriptive trace

Debug.Print "Desc: " & Err.Description & " | Source: " & Err.Source

End Function

## 🚀 SmartFilter & 🔄 SmartFilterRows

Locate and filter massive datasets instantly—**60% faster than AutoFilter**. Works on protected sheets, shared workbooks, and tables. Returns matching rows immediately—**no manual loops required**.

Public Sub SmartFilter\_Example()

Dim shTc As New AWE\_Sheet

Dim tcRowNbr As Variant

' ✅ Open and initialize the sheet in one step

shTc.Initialize "Timecard", 3

' ✅ Use Excel-like AutoFilter syntax—enhanced for protected and shared environments

shTc.SmartFilter "ProjectID", "=Prj-171"

shTc.SmartFilter "Date", ">=1/1/2021", "<=1/31/2021", xlAnd

' ✅ Count rows without relying on fragile SpecialCells logic

' Debug.Print "Matching Row Count: " & shTc.SmartFilterRows.Count

' ✅ Loop through rows confidently—even at Excel’s 1M+ row limit

For Each tcRowNbr In shTc.SmartFilterRows

' ✅ Access cells using friendly column names

' Debug.Print shTc.SmartCells(tcRowNbr, "Employee Name").Value & ", ROW#:" & tcRowNbr

Next tcRowNbr

' ✅ Clear the filter for reuse or to reset logic

shTc.SmartFilterClear

End Sub

# 🔐 Appendix A – Contract Summary

**Quick reference (at a glance) for ChatGPT’s required behavior when developing with AWE\_Sheet.**

**✅ Initialization & Discipline**

* Always open AWE\_Sheet.txt and verify function signatures, parameters, and return types before writing code.
* Do **not** infer or invent function behavior
* Use **only** public functions defined in the file
* Request clarification when signature intent is unclear

**📦 Function Usage**

* SmartRowGet → For multi-value reads
* SmartRowSet → Only when source/destination layouts match
* SmartCells → For single-value access or range references
* SmartFilter, SmartFilterRows, SmartFilterClear → For row filtering
* SmartLookup, SmartLookupRows, SmartLookupValues → For fast composite key searches

**🛠 Code Generation Rules**

* Use Set for ranges, objects, and collections
* Avoid divide-by-zero; validate inputs before using
* Use correct typing:
  + Long → Row numbers; Double → Calculations; Object → Dictionary or Range-based row results
* Use RaiseError for error handling, not Err.Raise

**🧾 Variable Naming Conventions**

| **Purpose** | **Pattern** | **Example** |
| --- | --- | --- |
| AWE\_Sheet object | sh<AbbrSheet> | shRes |
| Row number (Long) | <AbbrSheet>RowNbr | resRowNbr |
| Row range (Range) | <AbbrSheet>RowRng | resRowRng |
| Row dictionary (Object) | <AbbrSheet>RowDict | resRowDict |
| LookupMeta (Variant) | <AbbrSheet>Meta | resMeta |
| Filtered rows | <AbbrSheet>Fltr | resFltr |

💡 **Domain variables** are values that derive from sheet data (e.g., "ProjectID", "Revenue"), and must be prefixed by the **originating sheet** abbreviation—not the current sheet in use.

**🎯 Smart Matching Behavior**

* Composite keys must match **exact or wildcard**
* Empty columns are **not treated as wildcards**
  + ❌ SmartLookupRows "value"
  + ✅ SmartLookupRows "value", "\*"
* Keys must follow formatting defined in SrchCols

**🧠 Clarification Template – When User Instructions Are Vague**

Politely ask for missing elements:

* Context – What’s the task?
* Input – Sheet and header row
* Output – Where to write (if any)
* Join Logic – How sheets match
* Rules – Business logic (e.g., skip blanks, dedupe)
* Style – Developer level and comment expectations