|  |
| --- |
| HATCH MOTT MACDONALD |
| BOM Application Programming Standard |
| A Comprehensive Guide to HMM’s BOM Application for the VBA Programmer |
|  |
|  |
|  |

|  |
| --- |
|  |

**Contents**

[Introduction](#h.gjdgxs)

[Background and Evolution](#h.30j0zll)

[Breakdown of Complex Procedures](#h.1fob9te)

[Hiding/Unhiding Worksheets for “Enable Macro” Check](#h.3znysh7)

[Publishing RFP](#h.2et92p0)

[Issuing Client Summary](#h.tyjcwt)

[Automatic Sorting](#h.3dy6vkm)

[CADWorx Import](#h.1t3h5sf)

[Determining Quantity Deficit on a Per Site Basis](#h.4d34og8)

[Reordering Categories](#h.2s8eyo1)

[Procedure Index](#h.17dp8vu)

[Microsoft Excel Objects](#h.3rdcrjn)

[ADMIN](#h.26in1rg)

[BOLTSCHEDULE](#h.lnxbz9)

[INTRO\_MACROCHECK](#h.35nkun2)

[MATGRADE](#h.1ksv4uv)

[PIPESCHEDULE](#h.44sinio)

[ThisWorkbook](#h.2jxsxqh)

[VB\_CATEGORY](#h.z337ya)

[VB\_CHANGE\_LOG](#h.3j2qqm3)

[VB\_CLIENTINVENTORY](#h.1y810tw)

[VB\_COVERSHEET](#h.4i7ojhp)

[VB\_MASTER](#h.2xcytpi)

[VB\_MDR](#h.1ci93xb)

[VB\_ORDER\_LOG](#h.3whwml4)

[VB\_ORDER\_TMPLT](#h.2bn6wsx)

[VB\_RFP\_REPORT](#h.qsh70q)

[VB\_SITEBOM](#h.3as4poj)

[VB\_SITEDB](#h.1pxezwc)

[VB\_UNIQUEID](#h.49x2ik5)

[VB\_VAR\_STORE](#h.2p2csry)

[Modules](#h.147n2zr)

[CADWorx\_Module](#h.3o7alnk)

[EnableMacroCheck](#h.23ckvvd)

[ErrorHandling\_Module](#h.ihv636)

[ExcelExtensions](#h.32hioqz)

[ImportPrevious\_Module](#h.1hmsyys)

[MasterBOMMembers](#h.41mghml)

[Ribbon\_Module](#h.2grqrue)

[SiteBOMMembers](#h.vx1227)

[Classes](#h.3fwokq0)

[DynamicSiteBox](#h.1v1yuxt)

[The Custom RibbonUI](#h.4f1mdlm)

[BOM Development Techniques](#h.2u6wntf)

[Always use Code Names.](#h.19c6y18)

[Argument Referencing](#h.3tbugp1)

[Global Variables/Constants](#h.28h4qwu)

[RenderUI()](#h.nmf14n)

[DescriptionCompare()](#h.37m2jsg)

[Copying/Deleting Worksheets](#h.1mrcu09)

[LoadForm()](#h.2lwamvv)

[Debug.Print](#h.111kx3o)

[Option Explicit](#h.3l18frh)

[The Status Bar](#h.206ipza)

[Type Casting](#h.4k668n3)

[VBA Error Handling](#h.2zbgiuw)

[Online Resources](#h.1egqt2p)

[Definitions](#h.3ygebqi)

[Common Abbreviations](#h.2dlolyb)

[Thesaurus](#h.sqyw64)

[VBA Password](#h.3cqmetx)

# Introduction

Managing materials for Facilities can be a very complex effort and requires input from all design groups involved. Due to the tedious nature of managing the materials – from exporting a material list from a model, organizing an Excel based table, sending and maintaining Requests For Purchases with our clients, performing due-diligence on Engineering checks, and so on. In the past, many times data has been missed or lost and caused real challenges in maintaining a professional appearance to our clients. In order to try and streamline the process and put in measures to help prevent such mistakes in the future, HMM has assembled a tool for use by all design groups (Design/Drafting, Project Management, and Engineering), which will hopefully help mitigate some of the complex steps, and reduce the risks involved in manual data entry for this part of the design effort.

Most of the functionality for the BOM Application comes from code written in VBA. VBA stands for Visual Basic for Applications. It is a programming language based on Visual Basic that was developed by Microsoft to enable users to augment Microsoft Office documents. VBA procedures, also known as macros, are powerful and can be attached to forms, events, buttons, etc.

This programming standard is a go-by for future VBA development in the BOM Application. It is written with the intent that anyone with basic understanding of programming and the help of this guide and the internet can understand, update, and debug the VBA code for the BOM Application.

# Background and Evolution

Since HMM started developing a VBA element for the BOM Application, the terminology and design requirements have shifted slightly. The BOM Application was and always will be a living thing, evolving to meet the needs of those who use it.

#### In the beginning, there was nothing.

Matt Roberts handed me original version of this application. It had no VBA code. It only had the Master BOM list of items. Most of the functionality involved conditional formatting.

#### Inception

The first code that were added were most of the functions in ‘MasterBOMMembers’: procedures to add/delete rows/columns on the Master sheet, mark number/row number relationships, etc.

Early on there was also a strong desire for a Change Log. So the change logging was an integral part of the process from the start.

Order handling was the next thing to be added. In the beginning, RFPs were called RFQs, Orders, or Purchase Orders, and the RFP Form was called the Order Template. Some of the comments and function/variable names still refer to the older terminology like the code name for the RFP Form sheet: VB\_ORDER\_TMPLT.

#### Automatic Material Sorting

erg

#### Quality Control/Order Tracking/Inventory Managemen

Quality Control is critical to our process, but it’s dependent on a lot of different aspects of the process. Through the development process, this has gone through many generations.

The “Description Check” and “Delete?” columns on VB\_MASTER, quickly graduated to a double-click event to initiate the change and a comment on the cell where the double-click happened. Also, the Ribbon buttons that clear the Description Check and Quantity checks developed smoothly.

The issues started with verifying what material the Client had actually ordered and received. The client sometimes forgets to order material; sometimes material is lost or damaged in transit. Plus, different PMs like to handle these situations in different ways. Reconciling all the different possibilities and procedures without delving into detailed order tracking was a challenge.

As it stands, the “Client Inventory” column in VB\_MASTER represents the material supply the Client has/had on hand, exclusive of the material cataloged as RFPs. In other words, it’s a “catch all” for discrepancies to the RFP history. This could include material that the client had left over from another project, material that was lost or damaged in shipping, order mistakes, etc.

Order tracking in the BOM Application is just marking if the order has been received by the Client. That takes place in the *Order Manager*.

When mistakes are found in orders, things are broken, or the client finds or losses material, the *Inventory Manager* will reconcile these discrepancies. Unlike the “Client Inventory” column in VB\_MASTER, the *Inventory Manager* deals with the total material inventory to date, inclusive of everything. If the PM gets updates on the client’s total inventory to date, the *Inventory Manager* is the place to import that information, not the “Client Inventory” column. When the user is done with the *Inventory Manager*, it will back calculate the appropriate values for the “Client Inventory” column.

#### Model Discrepancy Reports

erg

#### Unique ID – CADWorx Plugin

CADWorx agreed to create a plug-in that interfaces with the BOM Application. The only requirement is that each material item needs to have a unique identifier for every material item: a “Unique ID”. Mark numbers are already unique identifiers, but once an item is purged from the list, that mark number becomes available again. The CADWorx plug-in required that the Unique IDs be singular, so that when a material item was purged, that Unique ID stayed unavailable.

The concept of Unique IDs is pretty fundamental and probably would have streamlined some aspects of development had this been implemented from the start, but this is how it evolved.

There is a hidden column on VB\_MASTER and a very-hidden sheet that assign and store Unique IDs. There is also a hidden column in VB\_CHANGE\_LOG for Unique IDs.

#### Consumables

#### Long Lead Items

erg

# Breakdown of Complex Procedures

## Hiding/Unhiding Worksheets for “Enable Macro” Check

erg

## Publishing RFP

Erg

## Issuing Client Summary

Erg

## Automatic Sorting

Erg

## CADWorx Import

Erg

## Determining Quantity Deficit on a Per Site Basis

Erg

## Reordering Categories

Erg

# Procedure Index

## Microsoft Excel Objects

rth

### ADMIN

erg

### BOLTSCHEDULE

erg

### INTRO\_MACROCHECK

Errrggg

### MATGRADE

Erg

### PIPESCHEDULE

Rth

### ThisWorkbook

Dfb

### VB\_CATEGORY

Erg

### VB\_CHANGE\_LOG

Erg

### VB\_CLIENTINVENTORY

Rth

### VB\_COVERS**H**EET

Rth

### VB\_MASTER

Erg

### VB\_MDR

Erg

### VB\_ORDER\_LOG

Erg

### VB\_ORDER\_TMPLT

Erg

### VB\_RFP\_REPORT

Erg

### VB\_SITEBOM

Th

### VB\_SITEDB

Erth

### VB\_UNIQUEID

Erth

### VB\_VAR\_STORE

Srth

## Modules

erg

### CADWorx\_Module

Wergf

### EnableMacroCheck

Erg

### ErrorHandling\_Module

Aerg

### ExcelExtensions

Erg

### ImportPrevious\_Module

Erg

### MasterBOMMembers

Erg

### Ribbon\_Module

Erg

### SiteBOMMembers

erg

## Classes

wef

### DynamicSiteBox

Wef

# The Custom RibbonUI

The Excel Ribbon customization was done in an application called “Custom UI Editor for Microsoft Office”. The editor is a free and easy to use application made for Office developers by Microsoft, not a third party. It is available from OpenXMLDeveloper.org – see Resources for a link. Only with this application can you open the Office document and view the custom xml.

In the editor, you append your own xml code to an existing Office document. For each document, there are two separate code pages for Office 2007 and for Office 2010+. For the most part, the xml code for each version of Office will be the same.

# BOM Development Techniques

There are important code snippets and techniques that help development from reducing runtime errors to maintaining the user interface. In the following subsections you’ll find these written out one by one. Some of these functions are ones that I have written, some not.

## Always use Code Names.

Every worksheet in Excel VBA has two names a sheet name and a code name. Sheet names are what the user sees on the sheet tabs in the user interface. Sheet names can be changed with code. Code names are static identifiers used in VBA for each sheet. They are read-only in VBA. You can use a code name as a worksheet object when coding. This makes code much easier to follow. For example if the codename for the “Master” worksheet is VB\_MASTER, then you can see in the following snippet that it yields simpler code.

ThisWorkbook.Worksheets(“Master”).Range(…)…

versus

VB\_MASTER.Range(…)…

Another obvious advantage to using codenames is that you don’t have to change your code when the sheet name changes. If the user changes a sheet’s name from “Master” to “MasterBOM”, and you haven’t been using codenames, your code will fail because Worksheets(“Master”) does not exist anymore. However, if you’ve employed a strict ‘only codenames’ policy, your code will continue to execute successfully.

The instances where codenames are not used in the BOM Application are for sheets that are creating or imported during runtime–when the codename is automatically generated by Excel and not set manually by a developer. In these cases, the code name could be anything, so it is safest to rename the sheet as soon as it’s created in the Workbook, then hide or protect it from the user so the name cannot be changed. With those precautions, you can very safely use the sheet name without fear of your code being broken.

## Argument Referencing

## Global Variables/Constants

Global variables are handy, but in VBA there are limitations and things to remember.

The largest scope for a constant is the entire project. These are public or private constants, but declared are the top of the object so it has scope within that whole object/project. Constants cannot be given a data type.

Public Const Err\_Exit = 0

Private Const first\_row = 5

The largest scope for a variable is also the entire project. These can be defined as a Public variable of any data type:

Public gblThis As Integer

Private ThisRibbon As IRibbonUI

The problem with global variables, is that on some runtime errors, when the VBA code breaks, global variable are reset to their Null values: 0 for Integers, False for Booleans, vbNullString for Strings, etc. Therefore, the smaller the scope is the better: avoid Public variables. Global variables and constants shall always be defined private along with Public Get/Set methods so you can access them from outside the object.

Standard Example:

Private gblThis As Integer

...

Public Function GetThis() As Integer

GetThis = gblThis

End Function

Public Sub SetThis(ByVal in\_this As Integer)

gblThis = in\_this

End Sub

Please ensure that each constant used is declared and defined once. Column numbers, reference cells, row numbers, etc shall all be declared as Private variables/constants so when changes are made to one of those constants, changes only need to be made in one part of the code.

## RenderUI()

When updating the user interface, disable Events and Screen Updating at the beginning of the procedure, and then enable them at the end. The will prevent your code from running slow and from starting events that don’t need to be run. It takes a lot of time for values to be painted on the screen. This should be done every time the code updates the user interface (UI).

Most of the time you see this done like this:

With Applcation

.EnableEvents = False

.ScreenUpdating = False

End With

...

With Applcation

.EnableEvents = True

.ScreenUpdating = True

End With

That’s fine if you are managing small bits of code. However, if the VBA code is involved where your multiple functions are disabling/enabling events all the time and/or simulatenously, it gets complicated.

Therefore I created the function RenderUI. This function will disable or enable both Events and ScreenUpdating at the same time. It will also return True or False telling the parent procedure whether or not the state changed. For example, the following code will render the UI False whether or not the UI state is already False. If the UI state was already False before RenderUI was called, ui\_change will be False.

Dim ui\_change As Boolean

ui\_change = RenderUI(False)

...

At the end of the same function, you’ll see this:

...

If ui\_change Then RenderUI True

This code ensures that the UI is rendered True only when ui\_change is True. In other words, only change the UI state when the UI state was changed at the beginning of the procedure.

Following this procedure will keep the UI state stable even when you have nested functions that all call RenderUI.

The only issue with using this is that RenderUI(True) is essentially like a repaint and will refresh everything on the screen. So many times you’ll get a screen flicker. During complicated procedures like CADImport where many cells in the UI are being updating, one screen flicker is nothing. However, if the user is manually entering values, a screen flicker every time a cell’s value is changed can get annoying.

## DescriptionCompare()

DescriptionCompare is a special kind of string comparison (StrComp) adapted for material descriptions. StrComp compares two Strings for equivalence. StrComp has only three return states:

-1 – for when the first String is ‘less than’ the second, alphabetically.

1 – for when the first String is ‘greater than’ the second, alphabetically.

0 – for when the first and second Strings are equal.

The issue with material is that two material descriptions may been equivalent, but return a nonzero value from StrComp. For instance:

* Numeric values could be in fractional or decimal form with or without leading/trailing zeros.
* Even though the information is the same, the punctuation between term may be different
* Special Keywords change the ordering such that ‘alphabetically’ isn’t a valid comparison tool.

DescriptionCompare handles all of these cases. It compares string word by word, skipping punctuation, evaluating decimal numbers and fractions as actual floating point numbers, not Strings. It also control the keywords “BLIND”, CONC”, “ECC”, “COATED”, and “BARE” to properly control the ‘less than’/‘greater than’ returns.

The ordering function of DescriptionCompare is the standard by which material items in VB\_MASTER are ordered within a category and appear on the screen.

Examples:

* DescriptionCompare( 'PIPE 1/2"', 'PIPE 0.500"' ) = 0
* DescriptionCompare( 'PIPE 0.28"', 'PIPE 0.280"' ) = 0
* DescriptionCompare( 'PIPE - 1/2"', 'PIPE, 1/4"' ) = 1
* DescriptionCompare( 'PIPE - 10", BARE', 'PIPE - 10", COATED' ) = 1
* DescriptionCompare( 'FLANGE, BLIND - 6"', 'FLANGE - 2"' ) = -1

## Copying/Deleting Worksheets

The thing to remember when coding a worksheet deletion/copy procedure is that the sheet must exist and the sheet must be visible, or you will get runtime errors. The safest way to do this is to

1. If the sheet doesn’t have a codename,
   1. create a worksheet object for the sheet and test to see if the worksheet exists,
2. remember the original visibility state
3. Set the sheet to Visible
4. Copy/Delete the sheet
5. Set the original sheet back to its original visibility, if applicable
6. Set the worksheet object = Nothing

...

Dim wrksht As Worksheet

On Error Resume Next

Set wrksht = ThisWorkbook.Worksheets(...)

Err.Clear

On Error Goto 0

If wrksht = Nothing Then

'Worksheet does not exist!

End If

Dim visState as Integer

visState = wrksht.Visibility

wrksht.Visibility = xlVisible

wrksht.Copy

...

wrksht.Visibility = visState

Set wrksht = Nothing

...

## LoadForm()

tyj

## Debug.Print

Rrrttth

## Option Explicit

wef

## The Status Bar

erg

## Type Casting

Erg

# VBA Error Handling

dfb

# Online Resources

Custom UI Editor for Microsoft Office

<http://msdn.microsoft.com/en-us/library/office/ee691832(v=office.14).aspx>

<http://openxmldeveloper.org/blog/b/openxmldeveloper/archive/2009/08/07/7293.aspx>

# Definitions

Argument

CADWorx Import

CADWorx Export

Category

Class

Codename

Event

Function

Return Value

Subroutine - a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

Procedure

Visual Basic for Applications

Module

User Form

Worksheet

Bill of Material

Request for Purchase

Consumable

Macro

String

Integer

Long

Double

Variant

Range

Name

## Common Abbreviations

BOM – Bill of Material

RFP – Request for Purchase

TMPLT - Template

VBA – Visual Basic for Applications

MDR – Model Discrepancy Report

VAR - Variable

XML

val - value

num - number

# Thesaurus

*Erg*

RFP

Order

RFP Template

RFP Form

RFP Client Report

Client Summary

Fab Package

Drafting Punchlist

Model Discrepancy Report

# VBA Password

zaq1@WSX