

USPS Roll Forward Model Excel/VBA
Program Documentation

Roll Forward Model Programmer's Manual

TABLE OF CONTENTS

I.	OVERVIEW.....	2
A.	Purpose of the Roll Forward Model.....	2
B.	Background and Model Update.....	2
II.	EXCEL/VBA MODEL DESIGN	3
A.	Capabilities.....	3
B.	Features	4
C.	Hardware.....	4
D.	Software	4
III.	VBA PROGRAM OVERVIEW.....	5
A.	Main Program (ModDriver).....	5
B.	Intialization Routines (ModInit)	5
C.	Calculations (modMAEffect).....	5
D.	Calculations (ModMEffect)	6
E.	Calculations (ModVEffect).....	7
F.	Calculations (ModAEEffect).....	7
G.	Piggybacks (ModIndirect).....	9
H.	Subtotaling Routines (ModSubtotals).....	9
I.	Error Checking Routines (ModErrorCheck)	9
J.	Other Initialization Routines - (ModInit)	10
K.	Utility Routines - (Module1 and Module 2)	10
IV.	FLOW DIAGRAM	13
V.	VBA PROGRAM DESCRIPTION.....	18
A.	Main Program (ModDriver).....	18
B.	Calculations (modMAEffect).....	20
C.	Calculations (ModMEffect)	22
D.	Calculations (ModVEffect).....	24
E.	Calculations (ModAEEffect).....	27
F.	Piggybacks (ModIndirect).....	35
G.	Subtotaling Routines (ModSubtotals).....	39
H.	Error Checking Routines (ModErrorCheck)	42
I.	Initialization Routines - (ModInit)	45
J.	Utility Routines - (Module1 and Module 2)	53
VI.	VBA PROGRAM CODE.....	67
A.	Main Program (ModDriver).....	67
B.	Calculations (modMAEffect).....	70
C.	Calculations (ModMEffect)	77
D.	Calculations (ModVEffect).....	84
E.	Calculations (ModAEEffect).....	92
F.	Piggybacks (ModIndirect).....	119
G.	Subtotaling Routines (ModSubtotals).....	138
H.	Error Checking Routines (ModErrorCheck)	144
I.	Initialization Routines - (ModInit)	152
J.	Utility Routines - (Module1 and Module 2)	172
K.	Roll Forward Distribution Key Loader Module.....	205
VII.	DIRECTORY OF GLOBAL VARIABLES	223

I. OVERVIEW

A. PURPOSE OF THE ROLLFORWARD MODEL

The roll forward model supports rate case filings by providing analyses of estimated accrued costs of the Postal Service by mail class or service for total future accrued costs, which form the basis for proposed changes in rates or fees. The analyses of estimated costs include an explanation of the effect on estimated total cost of projected cost level changes, projected mail volume levels, nonvolume workload factors, the change in the number of work days, the specification of the cost savings which will be realized from gains and improvements in total productivity, the identification of other program costs expected to be incurred in the forecasted test period (typically 2 to 3 years projected forward), and the workyear mix adjustment.

Costs are itemized by major categories or cost segments, such as postmasters, supervisors, city delivery, transportation, etc, and by individual cost elements, or components, within cost segments. The costs for the Postal Service are also categorized as those costs which can be attributed to a class of mail or type of mail service, and any costs which cannot be attributed. For R2005-1, the cost forecasting or "roll forward" (roll forward) model provides for each component the forecasting factors for seven different effects, including the workyear mix adjustment.

B. BACKGROUND AND MODEL UPDATE

The Postal Service has converted and redesigned its financial modeling system from a mainframe-based COBOL application to a PC-platform using off-the-shelf software. The driving factors for this redesign were to reduce the learning curve needed to use and run the model, to improve maintainability, to increase flexibility, and to give greater visibility to model actions. The new model has been developed with Microsoft Excel as the platform.

II. EXCEL/VBA MODEL DESIGN

A. CAPABILITIES

- Develops future costs by multiplying cost components by a series of factors
- Elements include:
 - cost matrix (input)
 - factors used to adjust cost matrix (input)
 - user specifications on how costs are to be adjusted (input)
 - effect workbook (output)
- Requires user-specified effects list. In R2005-1 there are 9 types of effects to choose from:
 - Cost Level Change: impact of resource price changes
 - Mail Volume Change: impact of change between current and prior years volumes
 - Non-Volume Workload: impact of change due to network or system-wide changes
 - Additional Workday: impact of change due to changes in number & type of days
 - Cost Reductions: impact of USPS programs resulting in cost savings.
 - Other Programs: impact of all other programs.
 - Corporate Wide Activities: impact of corporate wide activities.
 - Servicewide Costs: impact of service-wide costs.
 - Workyear Mix Adjustment: changes in mix of employees and overtime use.
- Run Options include:
 - Selected years (creates workbooks for all effects for each year)
- Sequence of Calculations:
 - Base Year cost matrix is input in the initial calculation.
 - Effect workbook generated by model includes a future cost matrix.
 - The cost matrix of the first effect calculated by the model is the input for the succeeding calculation (next effect on the list of effects)
 - The last effect cost matrix generated by the model incorporates cost changes due to all previous effects (as a result of sequential calculations)
 - Sequence of calculations corresponds to the order of effects specified by users in their list of effects.

B. BASIC FEATURES

- PC based program
- Microsoft Excel is the platform (Visual Basic for Applications is an integrated package)
- User specified effects parameters entered by users in model and in separate workbooks called Tables (one Table for each future year).
- CRA dictated parameters specified in Model
- Calculations are performed in effect workbooks (model generates formulas contained in workbooks based on user specifications).
- Effect workbooks generated by model are not linked. A new run is required to change results

C. HARDWARE

Recommend run on PC with Pentium IV with 256 Mb of RAM

D. SOFTWARE

Recommend MS Excel 2000 or MS Excel XP

III. VBA PROGRAM OVERVIEW

This section gives an overview of the RFModel.xls program, with a short description of the primary subroutines used in each main section of the program.

A. Main Program (ModDriver)

Subroutine Driver in module ModDriver is the main program loop for the roll forward model. It calls the initialization routines, and loops through the list of years and effects, calling the applicable calculation subroutine to implement each effect.

B. Initialization Routines (ModInit)

- Init (ModInit) - performs initializations used through program, reads in user-defined areas, reads in classes and components used to generate workbook template.
- ReadFileNames (ModInit) – reads in input table filenames and layouts and list of effects from user-defined areas.
- RefreshInputMatrix (Module1) – if user selects this option then this subroutine will read in the base year cost file and put it into the RFModel workbook.
- MakeBookNames (ModInit) – sets up array containing the names of the output workbooks that will be created during program execution.

C. Calculations (modMAEffect)

Subroutine MAEffect in module modMAEffect contains the code to generate output workbooks for type “MA” effects. The formulas in these workbooks will multiply all mail class costs in a component by a given factor. The subroutine creates and sets up the effect workbook, writes formulas, and generates the output costs matrix. MAEffect calls the following subroutines:

- GetInputMatrix (ModInit) – loads the prior scenario output costs as inputs for the new effect workbook
- GetClasses (Module2) – writes the mail class names onto each cost segment page in the effect workbook

- AddComps (Module1) – puts component names on the appropriate cost segment page in the effect workbook
- Template Formulas (modMAEffect) – develops formulas for type “MA” effects and writes the formulas into the effect workbook.
 - ComponentSubtotals (modSubtotals) – generates and writes formulas for subtotal components.
 - MakeOutputMatrix (Module1) – adds the costs for this effect to the input costs and creates an output cost matrix.

D. Calculations (ModMEffect)

Subroutine MEffect in module modMAEffect contains the code to generate output workbooks for type “M” effects. The formulas in these workbooks will multiply a given mail class costs in a component by a factor, or will multiply all mail class costs in a component by the factor. The subroutine creates and sets up the effect workbook, writes formulas, and generates the output costs matrix. MEffect calls the following subroutines:

- FindPreviousEffect (Module2) - finds the previous effect so that the costs can be read in
- AddCSSheet (Module2) – adds a worksheet for each cost segment to the effect workbook, which in turn calls GetClasses (Module1) to put the mail class names on the worksheet
- AddComps (Module1) – puts component names on the appropriate cost segment page in the effect workbook
- NVAWFormulas (modMEffect) – generates and writes the formulas into the effect workbook. These formulas will multiply one or all mail class costs by the given factor.
 - Piggybacks (ModIndirect) – generates and writes formulas to calculate indirect effects. The indirect effect is calculated by multiplying the cost for the indirect (piggyback) component by ratio of the costs after the effect has been applied to before the effect has been applied to the direct component(s).
 - ComponentSubtotals (ModSubtotals) – generates and writes formulas for subtotal components.

- **MakeOutputMatrix (Module1)** – adds the costs for this effect to the input costs and creates an output cost matrix.

E. Calculations (ModVEffect)

Subroutine VEffect in module modVEffect contains the code to generate output workbooks for type “V” effects. The formulas in these workbooks will multiply all mail class costs in a component by a vector of factors, one for each mail class. The subroutine creates and sets up the effect workbook, writes formulas, and generates the output costs matrix. VEffect calls the following subroutines:

- **FindPreviousEffect (Module2)** - finds the previous effect so that the costs can be read in
- **AddCSSheet (Module2)** – adds a worksheet for each cost segment to the effect workbook, which in turn calls **GetClasses (Module1)** to put the mail class names on the worksheet
- **AddComps (Module1)** – puts component names on the appropriate cost segment page in the effect workbook
- **MakeVolAdjFormulas (ModVEffect)** – generates and writes mail volume effect formulas into the effect workbook. There are different formulas depending on whether the mail volume effect is a regular mail volume effect or a volume mix adjustment effect.
- **Piggybacks (ModIndirect)** – generates and writes formulas to calculate indirect effects. The indirect effect is calculated by multiplying the cost for the indirect (piggyback) component by ratio of the costs after the effect has been applied to before the effect has been applied to the direct component(s).
- **MakeOutputMatrix (Module1)** – adds the costs for this effect to the input costs and creates an output cost matrix.

F. Calculations (ModAEffect)

Subroutine AEffect in module modAEffect contains the code to generate output workbooks for type “A” effects. The formulas in these workbooks will distribute an amount to mail classes and then add the distributed amounts to a given component. The subroutine creates and sets up the effect workbook, writes formulas, and

generates the output costs matrix. AEffect calls the following subroutines:

- CROPInit (ModAEffect) – reads in the data from the input table, giving the program amount, the distributing component, and the distribution component, and if used, the components to spread the cost to.
- CROPWorkbook (ModAEffect) – creates and sets up effect workbook, adds additional worksheets in the effect workbook to do calculations for each cost reduction or other program
 - AddCSSheet (Module2) – adds a worksheet for each cost segment to the effect workbook, which in turn calls GetClasses (Module1) to put the mail class names on the worksheet
 - AddComps (Module1) – puts component names on the appropriate cost segment page in the effect workbook
- CROPDist (ModAEffect) – generates and writes formulas to do cost reductions / other programs
 - Type1 (ModAEffect) – formulas for amounts distributed to mail classes by a component and added to the same component
 - Type2 (ModAEffect) – formulas where the amount is first spread among a group of components and then distributed to mail classes and added to each individual component
 - Type3 (ModAEffect) – formulas for amounts distributed to mail classes using one component and then added to another component
 - Type4 (ModAEffect) – formulas where the amount is first spread among a group of components and then distributed to mail classes using a given distribution key
- CROPSumtoCS (ModAEffect) – formulas to sum up all of the distributed amounts going to each component
 - Make Output Matrix (Module1) – adds the costs for this effect to the input costs and creates an output cost matrix.

G. Calculations (ModIndirect)

Subroutine Piggybacks in module modIndirect generates and writes formulas to calculate indirect effect costs. The subroutine first reads in the definitions of all distribution keys, which is the list of components used to calculate each indirect effect ratio. The subroutine next sets up two new worksheets in the given effect workbook. The first worksheet contains the costs for all components before the effect has been applied. The second worksheet contains the costs for all components after the effect is applied. The subroutine then cycles through the distribution key array in order and calls the following subroutines:

- GetRatios (modIndirect) – for the set of components in each distribution key, generates and writes a formula into the effect workbook that gives the percentage change by mail class before and after the effect has been applied.
- WriteIndirectFormula (modIndirect) – for each component getting an indirect effect, multiplies the input costs for each mail class in that component by the appropriate ratio. Writes formulas for both a regular indirect effect and an indirect mix effect.

H. Subtotaling Routines (modSubTotals)

This module contains the set of subroutines used to generate and write formulas to calculate mail class and component subtotals.

- ClassSubtotals (modSubTotals) – initializes an array with formulas to calculate the costs for subtotal classes. These formulas will be used in all effect types.
- InitClassRow (modSubTotals) – initializes an array containing the Excel row for each mail class.
- ComponentSubtotals (modSubTotals) - generates and writes formulas for subtotal components. The formula sums up the costs for each child component.

I. Error Checking Routines (modErrorCheck)

This module contains the subroutines that perform error checks on the input table data.

- ErrorCheckDriver – drives the error checking of the input tables

- `chkEffectList` – makes sure the `EffectsList` worksheet exists, then checks the data, making sure all expected data is there and that components and classes are valid.
- `chkA` – makes sure the `CostReductions`, `OtherPrograms`, and `WorkyearMix` worksheets exist, and then checks the data, making sure all expected data is there.
- `chkDK` – makes sure the `DistKeys` worksheet exists, and then checks the data, making sure distribution keys and addends are valid components.

J. Additional Initalizations (modlnit)

Section B lists initialization routines called by subroutine `Driver`. Additional initialization routines other than those mentioned in section B above are contained in this module.

- `DKInit (Modlnit)` – reads the distribution key definitions from the input table into an array.
- `Indirectlnit (Modlnit)` - called by the `piggybacks` subroutine to read the input table and fill in an array containing the components requiring an indirect effect.
- `InitializeWorkbookNames (Modlnit)` - populates an array for all possible years and all possible effects so that even previous year workbook names will be known.
- `FileCheck (Modlnit)` - verifies that all files that are needed for the run exist, including effects workbooks is the user does not check 'Create Workbooks as Needed', input table workbooks, and if the user selects 'Refresh Base Year' makes sure base year workbook exists.

K. Utilities

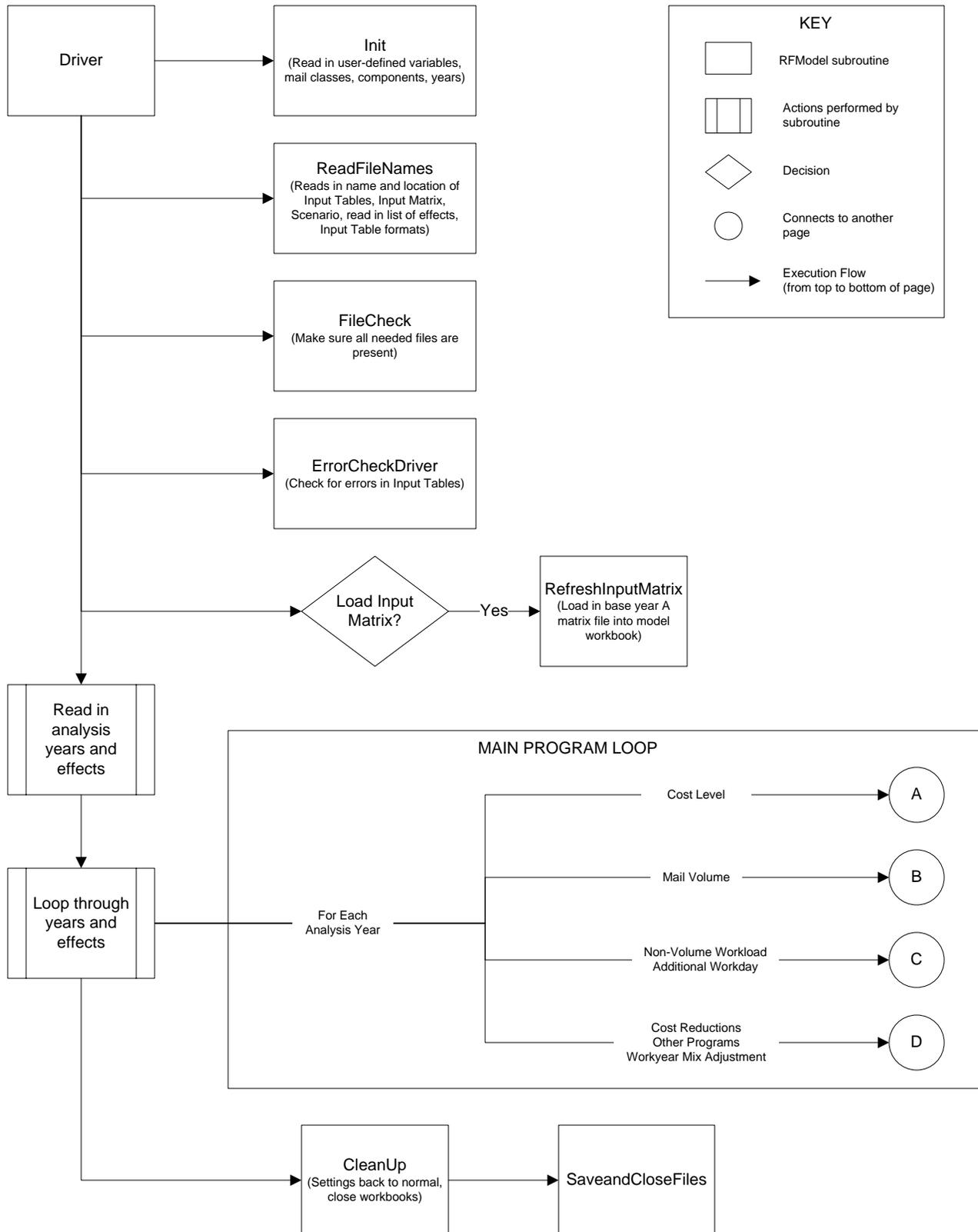
Two modules (`Module1` and `Module2`) contain utilities that are used throughout the program.

- `Cleanup (Module1)` – resets Excel settings when model completes execution.
- `SaveandCloseWorkbooks (Module1)` – saves and closes all opened workbooks and the model itself.

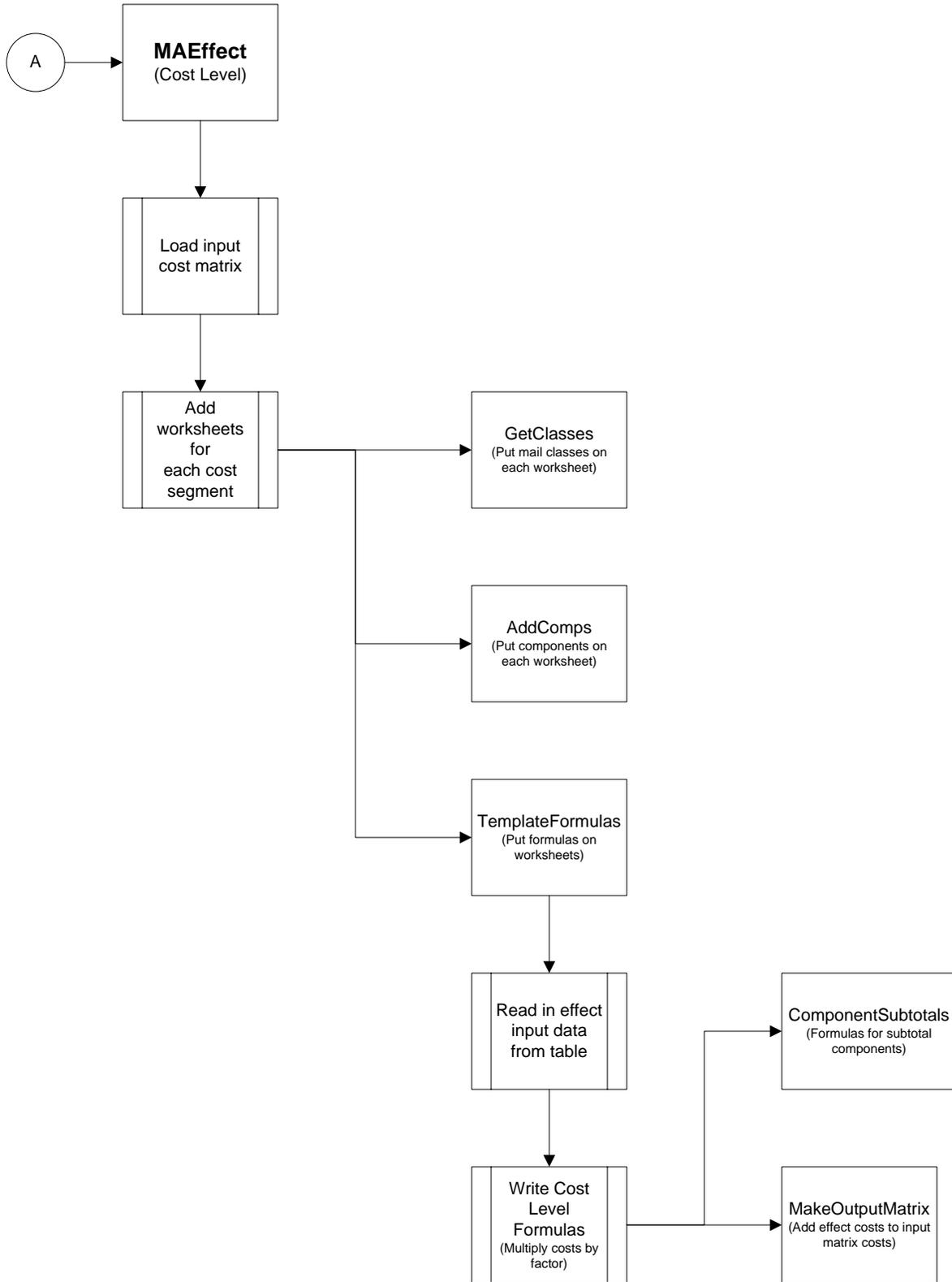
- RefreshInputMatrix (Module1) – reads in base year costs and writes into model.
- ClassArrayInit (Module1) – reads in the user-defined mail class information.
- GetClasses (Module1) – puts the mail class names and numbers onto the worksheet
- MakeOutputMatrix (Module1) – adds the costs for the effect by mail class and component to the input costs and generates an output matrix.
- Auto_Open (Module1) – runs automatically when the model is opened.
- AddComps (Module1) – puts component names onto the correct cost segment page in the effect workbook.
- OpenWB (Module2) – opens the given workbook in the given directory. Returns a code indicating whether the workbook already existed or whether it had to be created.
- AddCSSheet (Module2) – adds a page for each cost segment to the effect workbook. Calls GetClasses.
- FindChildren (Module2) – recursively finds the children for each parent component
- FindPreviousEffect (Module2) – finds the prior effect, which will contain the current effect's input costs
- IsClass (Module2) – makes sure a mail class is valid (i.e. is in the mailclasses array)
- IsComponent (Module2) – makes sure a component is valid (i.e. is in the components array)
- ScenarioChange (Module2) – updates the Scenario description when the user changes the scenario selection
- MultiYearOptionClick (Module2) – activated when user selects the multi-year radio button on the menu page.S
- SingleYearOptionClick (Module2) – activated when the user selects the single-year ratio button on the menu page.

- CheckEffectOrder (Module2) – checks to make sure the start effect the user selects on the single-year option is before the end effect.
- CheckYearOrder (Module2) – checks to make sure the start year the user selects on the multi-year option is before the ending year.
- UpdateLists (Module2) – updates all of the drop-down lists
 - UpdateScenarioList (Module2) – updates the scenarios drop-down
 - UpdateEffectsList (Module2) – updates the effects drop-down
 - UpdateYearsList (Module2) – updates the years drop-down

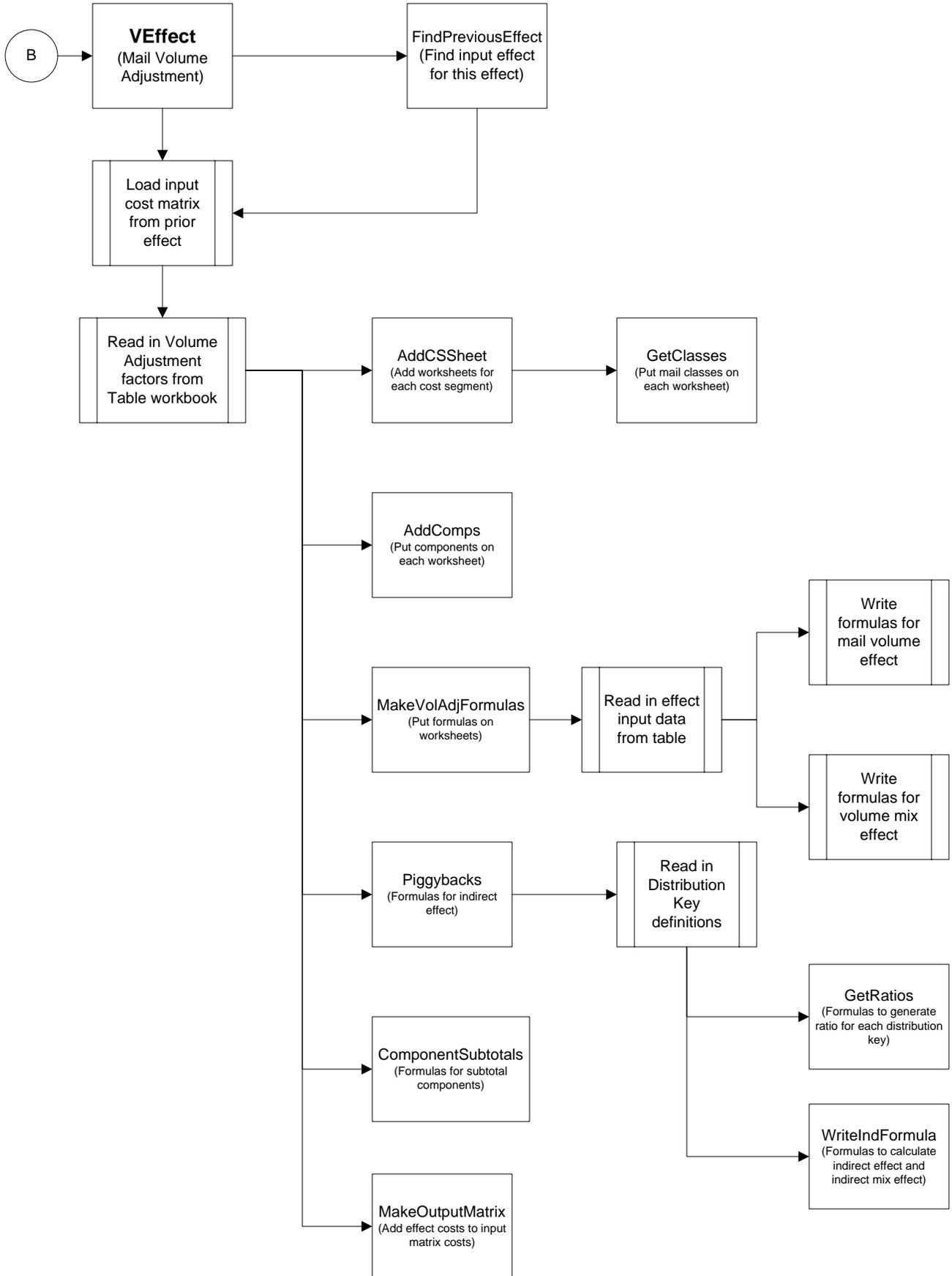
IV. FLOW DIAGRAM EXECUTION OF ROLL FORWARD MODEL



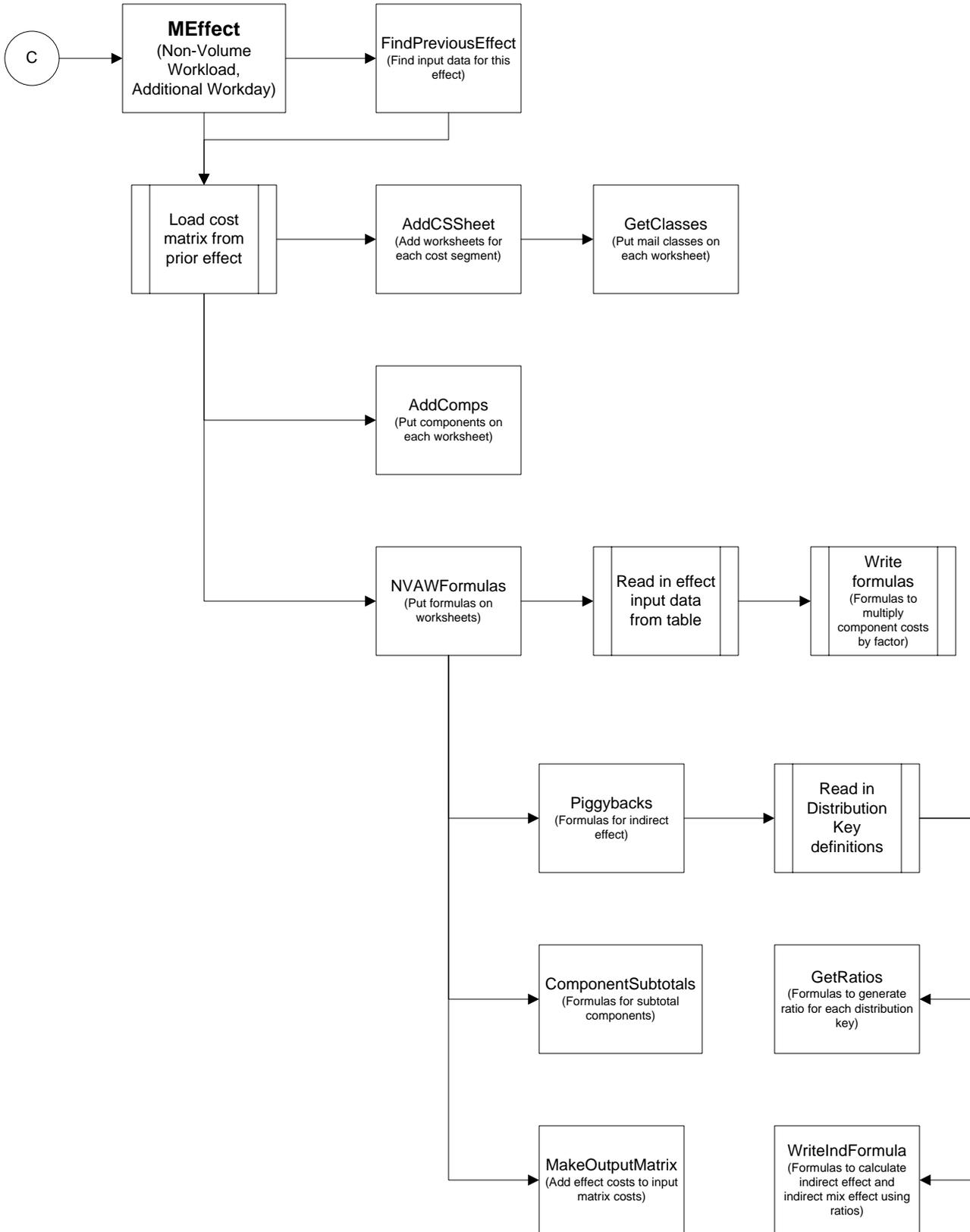
FLOW CHART FOR EXECUTION OF ROLL FORWARD MODEL - Page 14



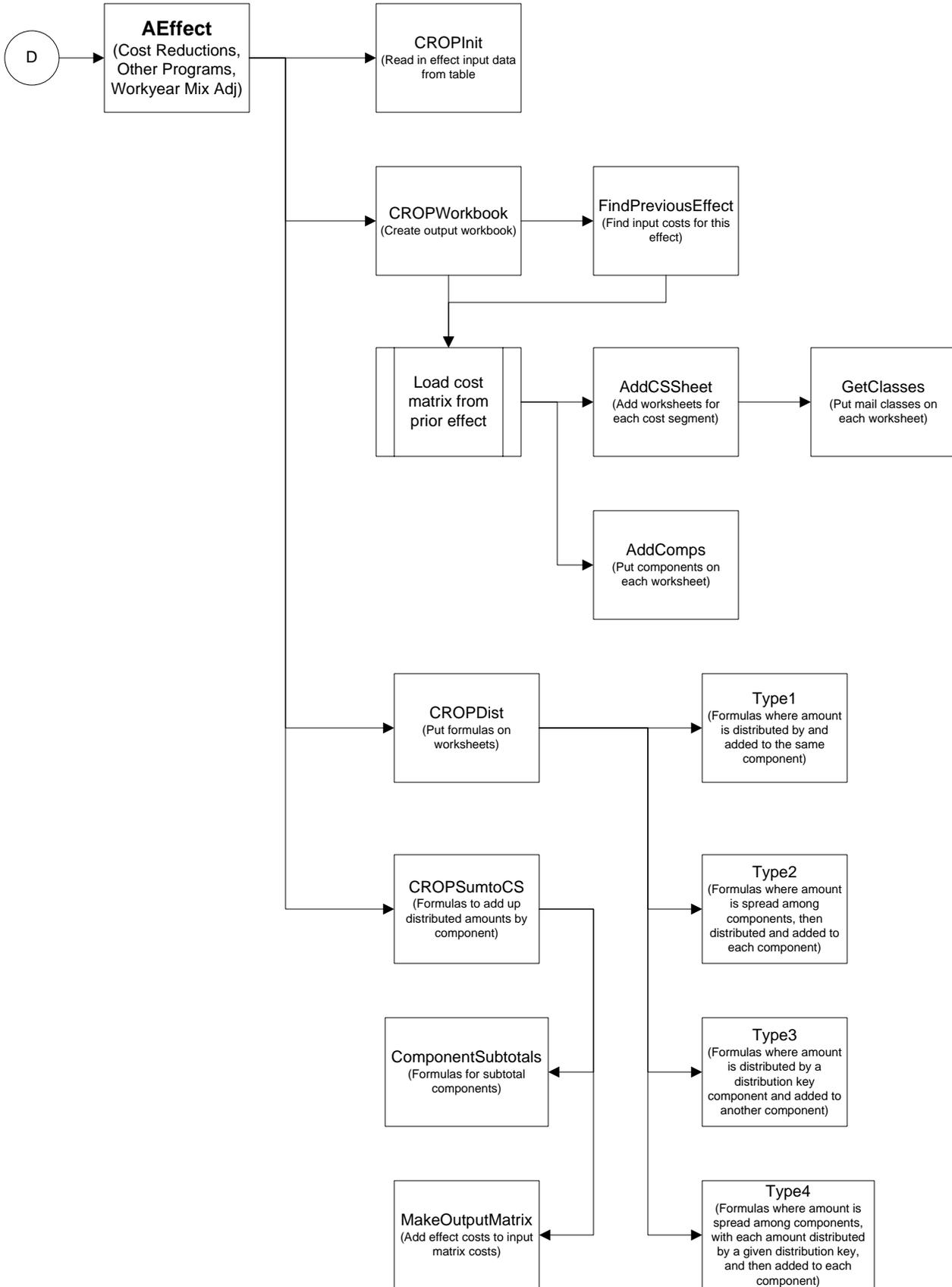
FLOW CHART FOR EXECUTION OF ROLL FORWARD MODEL - Page 15



FLOW CHART FOR EXECUTION OF ROLL FORWARD MODEL - Page 16



FLOW CHART FOR EXECUTION OF ROLL FORWARD MODEL - Page 17



V. VBA PROGRAM DESCRIPTION

This section describes each Visual Basic for Applications subroutine in RFModel.xls. The code for the subroutines are contained in program modules. A description of the purpose for each subroutine is shown, as well as the workbooks that are used as input, the global variables used as input, and if relevant, a description of the variables passed to this subroutine by another subroutine. Each section also lists the subroutines called by this subroutine, as well as the subroutines that call this subroutine. Finally, each section shows the workbooks that are changed or output by the subroutine and any global variables that are initialized.

A. Main Program - ModDriver

1. Subroutine: Driver()

Purpose:

- Executes the main program loop
- Calls initialization routines
- Verifies the user wants to overwrite existing workbooks, if applicable
- Calls FileCheck to make sure all needed files exist.
- Calls ErrorCheckDriver to verify data on input table workbooks.
- Calls GetInputMatrix to read in the base year input matrix and store it in the model, if applicable
- Runs the main program loop, which creates output workbooks for each effect in each selected year.
- Calls Cleanup to save and close output workbooks, if applicable, and reset Excel settings

Inputs:

Workbooks:

Base Year input matrix file
Input Table files for selected years
Roll Forward model – worksheets with user-defined inputs

Global Variables:

ynCreateWB
ynMultiYear
iStartYear
iEndYear
NumEffects
iYear
iStartEffect
iEndEffect
YearLabel array
PrevYearLabel array
EffectData array
ErrorCount

strErrorCheck

Outputs:

Workbooks:

Output workbooks for each roll forward effect in each selected year.

Initialized Global Variables:

IMatrixDir
IMatrixBook
strCurYear
strPrevYear
EffectsSheet
iEffect

Subroutines Called:

Init
FileCheck
ErrorCheckDriver
RefreshInputMatrix
MakeBookNames
MAEffect
MEffect
VEffect
AEEffect
Cleanup

Called By:

None

B. Calculations - ModMAEffect

1. Subroutine: MAEffect()

Purpose:

- Creates the output workbooks for all type "MA" effects
- Calls TemplateFormulas to build formulas for type "MA" effect

Inputs:

Workbooks:

Table for current year – Effects worksheet

Global Variables:

EffectData array
iEffect
DefaultDirectory
IMatrixSheet
ComponentSheet
MainBook
MaxCS
ScenDescription
strCurYear
EffectLabel
CSName array
StatusLabel

Outputs:

Workbooks:

Output workbook for this effect

Initialized Global Variables:

EffectLabel
StatusLabel

Subroutines Called:

GetInputMatrix
OpenWB
GetClasses
AddComps
ClassSubtotals
TemplateFormulas

Called By:

Driver

2. Subroutine: TemplateFormulas()

Purpose:

- Develops the formulas for type "MA" effects and writes them into the output workbook
- Reads Table workbook to get the factors associated with components
- Calls MakeOutputMatrix to create output cost matrix

Inputs:

Workbooks:

Table for current year

Global Variables:

InputWorkbook
InputDirectory
EffectsSheet
ECompon
ECompName
EffectData
iEffect
ComponentRow
SegCompon array
Classes array
ROffSet
COffSet
Row1
ClassFormulas array

Outputs:

Output workbook for this effect – formulas added as appropriate

Subroutines Called:

OpenWB
ComponentSubtotals
MakeOutputMatrix

Called By:

MAEffect

C. Calculations - ModMEffect

1. Subroutine: MEffect(NVAWMode)

Purpose:

- Creates the output workbooks for type "M" effect
- Calls NVAWFormulas to build formulas for type "M" effects

NOTE: Input parameter NVAWMode is not used

Inputs:

Workbooks:

Output workbook for prior effect, worksheet "OutputMatrix"

Global Variables:

EffectData array
iEffect
InputEffect
strCurYear
FYDirectory
FYIMatrix
CSSheetMode
EffectLabel

Outputs:

Output workbook for this effect

Initialized Global Variables:

StatusLabel
EffectLabel

Subroutines Called:

FindPreviousEffect
OpenWB
AddCSSheet
AddComps
NVAWFormulas

Called By:

Driver

2. Subroutine: NVAWFormulas(NVAWMode)

Purpose:

- Generates formulas for type "M" effects and writes formulas onto the output workbook
- Reads in factors from Table workbook Effects worksheet and places factors in output workbook
- Writes type of effect (All or specific class) for component on output workbook
- Calls Piggybacks to generate and write formulas for indirect components
- Calls MakeOutputMatrix to generate and write output cost matrix for effect into output workbook

Inputs:

Workbooks:

Table workbook for current year, worksheet Effects

Global Variables:

EffectData array
iEffect
EffectLabel
strCurYear
InputWorkbook
InputDirectory
EffectsSheet
ECompon
ECompName
Components array
SegCompon array
ComponentRow
MaxClass
Classes array
ClassFormulas array
ROffSet
COffSet
Row1

Outputs:

Output workbook for this effect – formulas added as appropriate

Initialized Global Variables:

EffectLabel
StatusLabel

Subroutines Called:

OpenWB
Piggybacks
ComponentSubtotals
MakeOutputMatrix

Called By:

MEffect

D. Calculations - ModVEffect

1. Subroutine: VEffect()

Purpose:

- Generates output workbooks for type "V" effects
- Creates output workbook
- Reads in vector of factors
- Calls MakeVolAdjFormulas to develop and write formulas into output workbook
- Calls Piggybacks to develop and write formulas for indirect components into output workbook
- Calls MakeOutputMatrix to calculate and write the output cost matrix

Inputs:

Workbooks:

Table workbook
Output workbook for prior effect, worksheet "OutputMatrix"

Global Variables:

EffectData array
iEffect
StatusLabel
strCurYear
InputWorkbook
InputDirectory
VolAdjSheet
VClassCol
VFactorCol
FYIMatrix
InputEffect
DefaultDirectory

Outputs:

Workbooks:

Output workbook for this effect

Initialized Global Variables:

VolAdjClass array
VolAdjFactor array
CSSheetMode
EffectLabel
StatusLabel

Subroutines Called:

OpenWB
FindPreviousEffect
AddCSSheet

AddComps
MakeAdjVolFormulas
Piggybacks
ComponentSubtotals
MakeOutputMatrix

Called By:

Driver

2. Subroutine: MakeAdjVolFormulas()

Purpose:

- Generates formulas for type "V" effect and writes onto output workbook
- Generates formulas for regular volume adjustment, where mail class costs multiplied by the corresponding adjustment factor
- Generates formulas for volume mix adjustment, where total component cost kept the same, and costs in mail classes adjusted based on current mail mix

Inputs:

Workbooks:

Table workbook, Effects worksheet

Global Variables:

InputWorkbook
EffectsSheet
ECompon
ECompName
EffectData array
iEffect
strCurYear
SegCompon array
ComponentRow
Row1
Col1
MaxClass
Classes array
ROffSet
COffSet
ClassFormulas array
ComponentOffset
ComponentName array
TotVClass

Outputs:

Output workbook for this effect – formulas added as appropriate

Subroutines Called:

None

Called By:

VEffect

E. Calculations - ModAEffect

1. Subroutine AEffect()

Purpose:

- Main control subroutine for the cost reductions, other programs, and workyear mix (Type "A") effects.
- Calls CROPInit to read in input data from Table
- Calls CROPWorkbook to create the output workbook
- Calls CROPDist to generate the formulas to do the distribution of each individual cost reduction or other program
- Calls CROPSumtoCS to generate formulas to sum up all of the cost reductions or other programs by component

Inputs:

None

Outputs:

None

Subroutines Called:

CROPInit
CROPWorkbook
CROPDist
CROPSumtoCS

Called By:

Driver

2. Subroutine CROPInit()

Purpose:

- Reads in the input data from the Table for the current year
- Determines type of cost reduction of other program
 - Type 1 = Program receiving amount is used to distribute amount
 - Type 2 = Amount first allocated to a group of components, then distributed using each component
 - Type 3 = Amount distributed using a distribution key component, and then added to the receiving component
 - Type 4 = Amount first allocated to a group of components, then distributed using a given component

Inputs:

Workbooks:

Table workbook, worksheet for this effect

Global Variables:

InputWorkbook
InputDirectory
iEffect
EffectData array
MaxCROP
CROPDefs array
CROPAmt

Outputs:

Initialized Global Variables:

CROPDefs array
CROPAmt
CROPName
CROPComp
CROPCompName
CROPDK
CROPDKType

Subroutines Called:

OpenWB

Called By:

AEffect

3. Subroutine CROPWorkbook()

Purpose:

- Creates the output workbook for the effect
- Adds worksheets to output workbook to do type1, type2, and type3 cost reductions or other programs

Inputs:

Workbooks:

Output workbook for prior effect

Global Variables:

EffectData array
InputEffect
iEffect
strCurYear
EffectLabel
FYDirectory
FYIMatrix
IMatrixSheet
CROPDefs array

strCurYear
EffectLabel
ScenDescription
ClassColumn
ComponentRow

Outputs:

Workbooks:

Output workbook for this effect

Initialized Global Variables:

EffectLabel
StatusLabel
IMatrixSheet
CSSheetMode

Subroutines Called:

FindPreviousEffect
OpenWB
AddCSSheet
AddComps
ClassSubtotals
GetClasses

Called By:

AEffect

4. Subroutine CROPDist()

Purpose:

- Cycles through input data array (CROPDefs) and calls the appropriate subroutine to generate the formulas for this cost reduction or other program
- Calls Type1 to distribute amounts using the receiving component
- Calls Type2 to allocate amounts to a group of receiving components and distribute each amount by the receiving component
- Calls Type3 to distribute amounts using a distribution key component

Inputs:

Global Variables:

CROPDefs array

Outputs:

None

Subroutines Called:

Type1
Type2
Type3

Called By:

AEffect

5. Subroutine Type1(Index)

Purpose:

- Processes all type 1 cost reductions or other programs
- Generates formulas and writes onto output workbook to distribute amount

Inputs:

Index – pointer in CROPDefs array for this cost reduction or other program

Global Variables:

CROPDefs array
ClassColumn
ComponentRow
ComponentOffset
ClassOffset
ClassColumn
Classes array
MaxClass
ROffSet
COffSet
ClassFormulas array

Outputs:

Workbooks:

Output workbook: Type 1 worksheet has formulas to distribute amount

Subroutines Called:

None

Called By:

CROPDist

6. Subroutine Type3(Index)

Purpose:

- Processes all type 3 cost reductions or other programs

- Generates formulas and writes onto output workbook to distribute amount

Inputs:

Index – pointer in CROPDefs array for this cost reduction or other program

Global Variables:

CROPDefs array
ClassColumn
ComponentRow
ComponentOffset
ClassOffset
Classes array
ROffSet
COffSet
TotClass
ClassFormulas array

Outputs:

Workbooks:

Output workbook: Type 3 worksheets have formulas to distribute amount

Subroutines Called:

None

Called By:

CROPDist

7. Subroutine Type2(Index)

Purpose:

- Processes all type 2 cost reductions or other programs
- Generates formulas and writes onto output workbook to allocate amount to a group of components
- Generates formulas and writes into output workbook to distribute allocated amounts to mail classes

Inputs:

Index – pointer in CROPDefs array for this cost reduction or other program

Workbooks:

Table workbook for current year

Global Variables:

CROPDefs array

EffectData array
iEffect
InputWorkbook
InputDirectory
CROPName
CROPDK
CROPDKType
FYDirectory
ClassColumn
ComponentRow
ComponentOffset
ROffSet
COffSet
TotClass
MaxClass
Classes array
ClassFormulas array

Outputs:

Workbooks:

Output workbook: Type 2 worksheets have formulas to allocate and distribute amount

InitializedGlobalVariables:

CROPAmt
CROPName
CROPComp
CROPCompName
CROPDK
CROPDKType

Subroutines Called:

FindChildren
OpenWB

Called By:

CROPDist

8. Subroutine Type4(Index)

Purpose:

- Processes all type 4 cost reductions or other programs
- Generates formulas and writes onto output workbook to allocate amount to a group of components
- Generates formulas and writes into output workbook to distribute allocated amounts to mail classes

Inputs:

Index – pointer in CROPDefs array for this cost reduction or other program

Workbooks:

Table workbook for current year

Global Variables:

CROPDefs array
EffectData array
iEffect
InputWorkbook
InputDirectory
CROPDK
FYDirectory
ClassColumn
ComponentRow
ComponentOffset
ROffSet
COffSet
TotClass
MaxClass
Classes array
ClassFormulas array

Outputs:

Workbooks:

Output workbook: Type 4 worksheets have formulas to allocate and distribute amount

InitializedGlobalVariables:

CROPAmt
CROPName
CROPDK

Subroutines Called:

FindChildren
OpenWB

Called By:

CROPDist

9. Subroutine CROPSumtoCS()

Purpose:

- Puts formulas on the cost segment pages of the output workbook to add up all of the distributed cost reductions or other programs for each component
- Calls MakeOutputMatrix to generate and write output cost matrix into output workbook

Inputs:

Workbooks:

Output workbook for this effect

Global Variables:

ComponentCount
Components array
LastCostCS
ComponentRow
ComponentOffset
CROPDefs array
ClassColumn
Classes array
MaxClass
ClassFormulas array

Outputs:

Workbooks:

Output workbook for this effect with formulas to sum up cost reductions or other programs to component

Subroutines Called:

ComponentSubtotals
MakeOutputMatrix

Called By:

AEffect

F. Piggybacks - ModIndirect

1. Subroutine Piggybacks()

Purpose:

- Sets up worksheets to be used in generating ratios for indirect effects
 - One worksheet (BEFF) contains all the costs before this effect has been applied, from the input cost matrix
 - Another worksheet (AEFF) contains all the costs after this effect has been applied, from the effect workbook.
- For each distribution key in the DKDefinitions array, sum the before effects costs for the components comprising the array, sum the after effects costs for the components comprising the key, generate a formula for this ratio for after effects / before effects. Write the formula for the ratio on the workbook page 'Ratios'.
- The indirect effect is the input cost matrix cost for the component times the appropriate ratio. Develop formula and write on cost segment pages of effect workbook.
- Calls DKInit to initialize the DKDefinitions array.
- Calls IndirectInit to read in the data for components requiring an indirect effect.
- Calls GetRatios to generate formulas for ratios and write into output workbook
- Calls WriteIndFormula to generate and write formulas on multiply costs from input cost matrix by ratios

Inputs:

Workbooks:

Output workbook for this effect.

Global Variables:

ScenDescription
strCurYear
EffectLabel
ComponentRow
ClassColumn
ComponentOffset
DKDefinitions array
ComponentName array
ClassOffset
Classes array
MaxClass
ROffSet
COffSet
ClassFormulas array
SegCompon array
MaxComps
IndirectFactor array
MaxDKComp
MaxClass

Outputs:

Workbooks:

Output workbook for this effect, with new worksheets AEFF and BEFF

Subroutines Called:

DKInit
IndirectInit
GetClasses
GetRatios
WriteIndFormula

Called By:

NVAWFormulas
VEffect

2. Subroutine GetRatios(IType, CompList, AddendCount, DKComponent)

Purpose:

- Called by main piggyback subroutine.
- Takes a given distribution key and generates a formula to give the percentage change by mail class for before and after the effect has been applied to the components comprising the distribution key
- For indirect mix effect, instead of a ratio generate a formula to add up all of the after effect costs only (which will be input matrix costs plus costs due to that effect on the cost segment pages in the output workbook)
- Writes the ratio formula onto the Ratios page of the output workbook

Inputs:

IType - contains the type of indirect effect (1 = regular, 2 = mix)
CompList – array of components comprising distribution key
AddendCount – the number of components in the CompList array
DKComponent – the distribution key's component number

Workbooks:

Output workbook for this effect

Global Variables:

ComponentRow
ComponentOffset
ClassColumn
ComponentName array
ClassOffset
MaxClass

Outputs:

Workbooks:

Formulas on Ratios page in output workbook for this effect

Subroutines Called:

None

Called By:

Piggybacks

3. Subroutine WriteIndFormula(IType, Compon, DKComponent)

Purpose:

- Called by main piggyback subroutine
- Writes formula to do indirect effect for the indirect component on the appropriate cost segment page of the output workbook. Uses the ratio on the Ratios page for the distribution key for this indirect component.
- For regular indirect effect, multiply input matrix costs (before effect) by the ratio
- For indirect mix effect, formula redistributes input matrix costs to reflect the distribution of the after effect costs for the components in the distribution key

Inputs:

IType - contains the type of indirect effect (1 = regular, 2 = mix)
Compon – The number for the component getting the indirect effect
DKComponent – the distribution key's component number

Workbooks:

Output workbook for this effect

Global Variables:

ComponentRow
ComponentOffset
ClassColumn
TotClass
SegCompon array
MaxClass
CompColumn
Classes array
ROffSet
COffSet
ClassFormulas array
OtherClass
TotVClass

Outputs:

Workbooks:

Output workbook for this effect, with indirect costs calculated

Subroutines Called:

None

Called By:

Piggybacks

G. Subtotaling Routines - ModSubtotals

1. Subroutine ClassSubtotals()

Purpose:

- Puts in formulas for mail classes that are subtotals of other mail classes
- Generates an array of subtotal formulas that are put in whenever these formulas are needed

Inputs:

Workbooks:

Output workbook for the effect calling the subroutine

Global Variables:

MaxClass
Classes array
ClassRow array

Outputs:

Initialized Global Variables:

ClassFormulas array

Subroutines Called:

InitClassRow

Called By:

MAEffect
CROPWorkbook

2. Subroutine InitClassRow()

Purpose:

- Initializes the ClassRow array with the row for each mail class in the output workbook
- Runs before the first time subtotals will be needed, i.e. at the end of the Cost Level effect before mail class subtotal formulas are written to the output workbook.

Inputs:

Workbooks:

Output workbook for the effect calling the subroutine

Global Variables:

MaxClass
ClassColumn
ClassOffset

Outputs:

Initialized Global Variables:

ClassRow array

Subroutines Called:

None

Called By:

ClassSubtotals
ComponentSubtotals

3. Subroutine ComponentSubtotals()

Purpose:

- Puts in formulas for components that are subtotals of other components
- Called by the subroutines for each effect type

Inputs:

Workbooks:

Output workbook for the effect calling the subroutine

Global Variables:

StatusLabel
MaxCompon
Components array
LastCostCS
ComponentRow
ComponentOffset
ComponentIndex array
MaxCompon
Components array
ClassRow array
ClassOffset
ClassColumn

Outputs:

Workbooks:

Output workbook for the effect calling the subroutine, with formulas in subtotal components

Subroutines Called:

InitClassRow

Called By:

TemplateFormulas
NVAWFormulas
VEffect
CROPSumtoCS

H. Error Checking Routines – modErrorCheck

1. Subroutine ErrorCheckDriver()

Purpose:

- For each year selected for run, drives checking of input table workbooks
- Calls chkEffectList to check EffectsList worksheet
- Calls chkA to check CostReductions, Other Programs, and WorkyearMix worksheets
- Calls chkDK to check DistKeys worksheet

Inputs:

Workbooks:

Input Table workbooks

Global Variables:

ynMultiYear
iStartYear
iEndYear
NumEffects
iYear
iStartEffect
iEndEffect
InputTablesFile
DefaultDirectory
EffectData array

Outputs:

Initialized Global Variables:

ErrorCount
strErrorCheck

Subroutines Called:

chkEffectList
chkA
chkDK

Called By:

Driver

2. Subroutine chkEffectList(eFlg)

Purpose:

- Performs error checks on worksheet EffectsList
- First makes sure the worksheet exists in the given input table
- Makes sure component numbers are valid

- Makes sure cost level factors are either numeric or blank
- Makes sure mail volume method is either V, M, I, or IM, or blank and makes sure a valid distribution key exists for each I entry
- Makes sure nonvolume workload and additional workday factors are numeric or blank, and that the Method column contains a valid component number of an A

Inputs:

Workbooks:

Input table workbooks

Variables:

eFlg – indicates if first run through for this input table.

Global Variables:

EffectData array
ECompon

Outputs:

Initialized Global Variables:

ErrorCount
strErrorCheck

User-Defined Functions Called:

IsComponent
IsClass

Called By:

ErrorCheckDriver

3. Subroutine chkA()

Purpose:

- Makes sure the CostReductions, OtherPrograms, and Workyear Mix worksheets exist in the input table workbook.
- Checks to make sure valid data on worksheets

Inputs:

Workbooks:

Input table workbooks

Global Variables:

EffectData array

Outputs:

Initialized Global Variables:

ErrorCount
strErrorCheck

User-Defined Functions Called:

IsComponent

Called By:

ErrorCheckDriver

4. Subroutine chkDK()

Purpose:

- Makes sure the DistKeys worksheet exists
- Checks entries for valid data

Inputs:

Workbooks:

Input table workbooks

Global Variables:

DistKeysSheet
DKNum
UsedinRF
AddendNum

Outputs:

Initialized Global Variables:

ErrorCount
strErrorCheck

User-Defined Functions Called:

IsComponent

Called By:

ErrorCheckDriver

I. Initialization Routines - ModInit

1. Subroutine Init()

Purpose:

- Initializes global variables with user-defined options, including base year, test year, scenario to run, USPS or PRC run, column and row offsets for input matrix, multi-year or single year run.
- Calls ReadFileNames to read in names of input files from user-defined area.
- Reads in user-defined limits used to dimension arrays and then dimensions public arrays.
- Reads in data from ComponentsUSPS or ComponentsPRC worksheet in model
- Calls ClassArrayInit to read in user-defined mail class information from Classes worksheet in model
- Calls CSNameInit to read in user-defined list of cost segment numbers and associated names from CostSegments worksheet in model
- Calls InitializeWorkbookNames to set up an array containing the list of all effect workbooks that will be created during the model run.

Inputs:

Workbooks:

Roll forward model – worksheets Menu, EffectsParameters, Limits, ComponentsUSPS or ComponentsPRC, MailClasses, CostSegments.

Global Variables:

SCCount

Outputs:

Initialized global variables:

Col1
Row1
ComponentOffset
ClassColumn
ComponentRow
ClassOffset
MainBook
MainSheet
SaveCloseNames array
SCCount

Initialized using inputs from RFModel worksheet Menu:

BaseYear
TestYear
ScenarioRow
USPSFlag
ComponentSheet
ynMultiYear
ynCreateWB

ynSaveClose

Initialized using inputs from RFModel worksheet InputWorkbooks

ROffSet
COffSet

Initialized using inputs from RFModel worksheet Limits:

MaxClass
MaxCS
MaxCompon
MaxEffects
MaxYears
InputClassStart
MaxDKComp
MaxComps
MaxDefs
TotVClass
OtherClass
TotClass
MaxCROP
Max2Comp
LastCostCS

Initialized using inputs from RFModel Components worksheet:

SegCompon array
ComponentName array
ComponentIndex array
Components array
ComponentCount

Subroutines Called:

ReadFileNames
ClassArrayInit
CSNameInit
InitializeWorkbookNames

Called By:

Driver

2. Subroutine ReadFileNames(ErrFlag)

Purpose:

- Sets global variables from user-defined areas, including scenario name, description, and directory, input matrix worksheet name, distribution key worksheet name, volume adjustment worksheet name
- Reads in format for distribution key worksheet in Input Table file, format for effects worksheet in Input Table file
- Read in the list of effects
- Reads in the names of the Input Table files
- Reads in the analysis start and end year or the analysis start and end effect.
- Returns ErrFlag = 1 if default directory left blank

Inputs:

Workbooks:

RFModel - worksheets with user-defined inputs.

Global Variables:

ScenarioRow
ynMultiYear

Outputs:

Initialized Global Variables:

Initialized using RFModel inputs on worksheet Scenarios:

ScenName
ScenDescription
ScenWB
DefaultDirectory
InputDirectory

Initialized using RFModel inputs on worksheet InputWorkbooks:

IMatrixDir
IMatrixBook
IMatrixSheet
IMatrixCheck
IMatrixStart

Initialized using RFModel inputs on worksheet EffectsParameters:

DistKeysSheet
FYIMatrix
VolAdjSheet
UseInRF
DKNum
AddendNum
ECompon
ECompName
VClassCol
VFactorCol
NumEffects
EffectData array

Initialized using inputs on RFModel worksheet InputWorkbooks:

YearLabel array
InputTablesFile array
PrevYearLabel array

Initialized from RFModel worksheet Menu:

iStartYear
iEndYear

iYear
iStartEffect
iEndEffect

Subroutines Called:

None

Called By:

Init

3. Subroutine CSNameInit()

Purpose:

- Initializes CSName array with the name and title of each cost segment

Inputs:

Workbooks:

Roll forward model - worksheet CostSegments.

Global Variables:

MaxCS

Outputs:

Initialized Global Variables using RFModel inputs on worksheet CostSegments

CSName array

Subroutines Called:

None

Called By:

Init

4. Subroutine DKInit()

Purpose:

- Initializes array containing information on distribution keys

Inputs:

Workbooks:

Table worksheet DistKeys

Global Variables:

InputWorkbook
InputDirectory
DistKeysSheet
DKNum
AddendNum

Outputs:

Initialized global variables from data on Table worksheet DistKeys:

DKDefinitions array

Subroutines Called:

OpenWB

Called By:

Piggybacks

5. Subroutine FileCheck()

Purpose:

- Called by subroutine Driver to verify that all files needed for the run exist.

Inputs:

Global variables:

ynMultiYear
iStartYear
iEndYear
NumEffects
iYear
iStartEffect
iEndEffect
DefaultDirectory
InputTablesFile array
ynCreateWB
AllWorkbookNames array

Outputs:

None

Subroutines Called:

None

Called By:

Driver

6. Subroutine GetInputMatrix()

Purpose:

- Copies the base year input cost matrix from the RFmodel workbook into the output workbook for the effect.
- If this is the first effect for the 2nd or 3rd future year then copies the output cost matrix from the designated effect in the prior year

Inputs:

Workbooks:

Base year input matrix workbook

Global variables:

iYear
iEffect
iMatrixSheet
iMatrixStart
MainBook
iMatrixBook
iMatrixDir
TargetBook
FYIMatrix

Outputs:

Workbooks:

RFmodel - Base year input cost matrix written into workbook.

Initialized Global Variables:

IMatrixSheet

Subroutines Called:

OpenWB

Called By:

MAEffect

7. Subroutine IndirectInit()

Purpose:

- Called by main piggyback subroutine to read in data from the Table worksheet Effect. Data indicates if a component gets an indirect effect and the distribution key for that component getting an indirect effect.

Inputs:

Workbooks:

Input Table worksheet 'Effects'

Global variables:

InputWorkbook
InputDirectory
EffectsSheet
ECompon
ECompName
MaxComps
EffectData array
DKNum

Outputs:

Initialized Global Variables using Table worksheet Effects:

IndirectFactor array
DistributionKey array

Subroutines Called:

OpenWB

Called By:

Piggybacks

8. Subroutine InitializeWorkbookNames()

Purpose:

- Populates an array with the names of the workbooks that could possibly be created for all possible effects and all possible years, so that even prior year workbook names will be known.

Inputs:

Global variables:

TotalYears
TotalEffects
YearLabel array
EffectData array
ScenWB

Outputs:

Initialized Global Variables:

AllWorkBookNames

Subroutines Called:

None

Called By:

Init

9. Subroutine MakeBookNames()

Purpose:

- Constructs all the appropriate workbook names for the scenario and effects in the current year.

Inputs:

Global variables:

iYear
strPrevYear
strCurrYear
EffectData array
ScenWB
NumEffects

Outputs:

Global variables:

IMatrixBook
InputWorkbook
EffectData array element WorkbookName

Subroutines Called:

None

Called By:

Driver

J. Utility Routines – Module1 and Module2

1. Subroutine AddComps()

Purpose:

- Puts component names and numbers onto the output workbook
- Components in order they appear in RFModel Component worksheet

Inputs:

Workbooks:

Output workbook for the effect calling the subroutine

Global Variables:

ComponentCount
Components array
ComponentRow
LastCostCS
ComponentOffset

Outputs:

Workbooks:

Output workbook with component names and numbers placed on worksheet pages.

Subroutines Called:

None

Called By:

VEffect
CROPWorkbook
MAEffect
MEffect

2. Subroutine Auto_Open()

Purpose:

- Runs automatically when the workbook opens
- Activates Menu page and sizes to screen
- Updates scenario, effects, and year lists used in drop-downs
- Writes version number on Menu page
- Calls UpdateScenarioList, UpdateEffectsList, UpdateYearsList

Inputs:

Workbooks:

RF Model workbook

Outputs:

None

Subroutines Called:

UpdateScenarioList
UpdateEffectsLists
UpdateYearsLists

Called By:

None

3. Subroutine ClassArrayInit()

Purpose:

- Initializes the MailClasses array from the MailClasses page in the RFModel workbook

Inputs:

Workbooks:

RFModel workbook

Outputs:

Initialized Global Variables:

Classes array
ClassMax

Subroutines Called:

None

Called By:

Init

4. Subroutine Cleanup()

Purpose:

- Called when program completes execution
- Calls SaveCloseWorkbooks to save and close any opened workbooks
- Saves RFModel workbook
- Settings changed by model reset

Inputs:

Workbooks:

None

Global Variables:

MainBook

Outputs:

Workbooks:

Output workbooks saved and closed
RFModel saved, any settings changed by model reset

Subroutines Called:

SaveCloseWorkbooks

Called By:

Driver

5. Subroutine FindText(Txt As String)

Purpose:

- NOT CURRENTLY USED

6. Subroutine GetClasses()

Purpose:

- Puts the mail classes onto the current worksheet in the output workbook

Inputs:

Workbooks:

Current worksheet in output workbook

Global Variables:

ClassMax
Classes array
CSSheetMode
MaxClass
VolAdjClass array
VolAdjFactor array
Row1
Col1

CSSheet

Outputs:

Workbooks:

Current worksheet in output workbook with mail class names and numbers

Subroutines Called:

None

Called By:

AddCSSheet
Piggybacks
CROPWorkbook
MAEffect

7. Subroutine MakeOutputMatrix()

Purpose:

- Generates an output cost matrix in the open output workbook
- Deletes OutputMatrix if already exists (workbook already existed from prior run)
- Adds costs from costs for this effect to input matrix costs

Inputs:

Workbooks:

Output workbook for the effect calling the subroutine

Global Variables:

IMatrixSheet
StatusLabel
InputClassStart
ROffSet
COffSet
Col1
Row1
CSSheetMode

Outputs:

Output workbook with OutputMatrix worksheet added

Subroutines Called:

None

Called By:

VEffect

CROPSumtoCS
TemplateFormulas
NVAWFormulas

8. Subroutine RefreshInputMatrix()

Purpose:

- Copies base year input cost matrix into RFModel

Inputs:

Workbooks:

Base year cost matrix workbook
RFModel workbook

Global Variables:

IMatrixDir
IMatrixBook
IMatrixSheet
IMatrixCheck
ROffSet
COffSet
IMatrixStart

Outputs:

Workbooks:

RFModel workbook InputMatrix worksheet updated

Subroutines Called:

OpenWB

Called By:

Driver

9. Subroutine SaveCloseWorkbooks()

Purpose:

- Saves and closes any workbooks that the model run opened or created
- Called by subroutine Cleanup

Inputs:

Workbooks:

All workbooks opened and created by model run

Global Variables:

ynSaveClose
SCCount
SaveCloseNames array

Outputs:

Workbooks:

All workbooks opened by model are closed

Subroutines Called:

None

Called By:

Cleanup

10. Subroutine AddCSSheet()

Purpose:

- Adds cost segment worksheets to opened output workbook
- Gets cost segments to add from RFModel Components worksheet
- Calls GetClasses to put mail class names and numbers on worksheet
- Adds 'Ratios' worksheet for use in indirect costs

Inputs:

Workbooks:

Opened output workbook
RFModel workbook Components worksheet

Global Variables:

ComponentSheet
MainBook
MaxCS
CSName array
NumEffects
EffectData array
iEffect
ScenDescription
strCurYear
EffectLabel
iEffect

Outputs:

Workbooks:

Cost segment pages added to output workbook

Initialized Global Variables:

CSSheet

Subroutines Called:

GetClasses

Called By:

VEffect
CROPWorkbook
MEffect

11. Subroutine CheckEffectOrder()

Purpose:

- Used when user selects analysis option to run a selection of effects in a single year
- Makes sure start effect is before end effect

Inputs:

Workbooks:

RFModel workbook

Outputs:

Initialized Global Variables:

iStartEffect
iEndEffect

Subroutines Called:

None

Called By:

None

12. Subroutine CheckYearOrder()

Purpose:

- If user selects option to run multiple years, makes sure start year is before end year

Inputs:

Workbooks:

RFModel

Outputs:

Initialized Global Variables:

iStartYear
iEndYear

Subroutines Called:

None

Called By:

None

13. Subroutine FindChildren(Cmpt, FCMode As Integer, OutArray)

Purpose:

- Finds all the child components of a given component

Inputs:

Cmpt – the component to find the children of
FCMode – 1 indicates find immediate children, 2 indicates find all descendents
OutArray – declared by calling routine, holds the children components of Cmpt

Global Variables:

ComponentCount
Components array

Outputs:

Initialized Variables:

OutArray

Subroutines Called:

None

Called By:

Type2
Type4

14. Subroutine FindPreviousEffect()

Purpose:

- Finds the previous effect whose output matrix is the current effect's input matrix

Inputs:

Global Variables:

EffectData array
iEffect
NumEffects

Outputs:

Initialized Global Variables:

InputEffect

Subroutines Called:

None

Called By:

VEffect
CROPWorkbook
MEffect

15. Subroutine FindPreviousEffect()

Purpose:

- Finds the previous effect whose output matrix is the current effect's input matrix

Inputs:

Global Variables:

EffectData array
iEffect
NumEffects

Outputs:

Initialized Global Variables:

InputEffect

Subroutines Called:

None

Called By:

VEffect
CROPWorkbook
MEffect

16. Public Function IsClass(C)

Purpose:

- Verifies a given class C is in the Classes array and is not empty

Inputs:

Global Variables:

Classes
MaxClass

Outputs:

Returns TRUE or FALSE

Subroutines Called:

None

Called By:

CheckEffectList

17. Public Function IsComponent(C)

Purpose:

- Makes sure given component C is in the Components array

Inputs:

Global Variables:

Components array

Outputs:

Returns TRUE or FALSE

Subroutines Called:

None

Called By:

CheckEffectList
chkA
chkDK

18. Subroutine OpenWB(OpenBook, OpenPath, Result As Integer)

Purpose:

- Opens workbook specified in input arguments
- Returns result code indicating:
 - 0 = workbook was already open before calling
 - 1 = workbook already existed, and is now open
 - 2 = workbook did not exist and had to be created

Inputs:

OpenBook – name of workbook
OpenPath – directory of workbook

Global Variables:

DefaultDirectory
strCurYear

Outputs:

Workbooks:

Workbook opened

Initialized Variables:

Result
SaveCloseNames array
SCCCount

Initialized Global Variables:

SCCCount
SaveCloseNames array

Subroutines Called:

None

Called By:

VEffect
CROPInit
CROPWorkbook
Type2
Type4
MAEffect
TemplateFormulas
DKInit
IndirectInit
GetInputMatrix
MEffect
NVAWFormulas
RefreshInputMatrix

19. Subroutine ScenarioChange()

Purpose:

- Updates scenario description on Menu page when user changes scenario selection using drop-down

Inputs:

Workbooks:

RFModel workbook

Outputs:

Initialized Global Variables:

ScenarioRow

Subroutines Called:

None

Called By:

None

20. Subroutine SingleYearOptionClick()

Purpose:

- Runs when the user clicks the single-year option button

Inputs:

Workbooks:

RFModel workbook

Outputs:

Initialized Global Variables:

ynMultiYear

Subroutines Called:

None

Called By:

None

21. Subroutine UpdateEffectsLists()

Purpose:

- Called by subroutine UpdateLists to update drop-down with list of effects

Inputs:

Workbooks:

RFModel workbook

Outputs:

Initialized Global Variables:

TotalEffects

Subroutines Called:

None

Called By:

Auto_Open
UpdateLists

22. Subroutine UpdateLists()

Purpose:

- Runs when user presses button to update lists
- Calls UpdateScenarioList, UpdateEffectsLists, UpdateYearsLists

Inputs:

Workbooks:

RFModel workbook

Outputs:

None

Subroutines Called:

UpdateScenarioList
UpdateEffectsLists
UpdateYearsLists

Called By:

None

23. Subroutine UpdateScenarioList()

Purpose:

- Called by subroutine UpdateLists to update drop-down with list of valid scenarios

Inputs:

Workbooks:

RFModel workbook

Outputs:

None

Subroutines Called:

None

Called By:

Auto_Open
UpdateLists

24. Subroutine UpdateYearsLists()

Purpose:

- Called by subroutine UpdateLists to update drop-down with list of years

Inputs:

Workbooks:

RFModel workbook

Outputs:

Initialized Global Variables:

TotalYears

Subroutines Called:

None

Called By:

Auto_Open
UpdateLists

VI. VBA PROGRAM CODE

A. Main Program - ModDriver

Sub Driver()

```
'Drives the main loop for the program
Dim e1, e2 As Integer
Dim y1, y2 As Integer
Dim InitErr, InitSep, InitTitle As String
Dim il As Long

'Initialization routine
Eflag = 0
Call Init
'Eflag is set to 1 for a variety of error conditions which require the run be aborted
If Eflag = 1 Then
Workbooks(ThisBook).Activate
Application.GoTo Reference:="ScenariIndex"
Application.ScreenUpdating = True
Exit Sub
End If

'Make sure all the input files needed for the run are present
Call FileCheck
If Eflag > 0 Then
Workbooks(ThisBook).Activate
Application.GoTo Reference:="ScenariIndex"
Application.ScreenUpdating = True
Application.StatusBar = False
Exit Sub
End If

'Make sure data in the input files is in the correct format
Call ErrorCheckDriver
If ErrorCount > 0 Then
InitErr = "The following input data error"
InitTitle = "Initialization Input Error"
If ErrorCount > 1 Then
InitErr = InitErr & "s were found:"
InitTitle = InitTitle & "s"
Else
InitErr = InitErr & " was found: "
End If
InitSep = vbCrLf & vbCrLf
For il = 1 To ErrorCount
InitErr = InitErr & InitSep & strErrorCheck(il)
Next
InitErr = InitErr & InitSep & "Processing cannot continue."
MsgBox InitErr, vbOKOnly + vbCritical, InitTitle
Workbooks(ThisBook).Activate
Application.GoTo Reference:="ScenariIndex"
Application.ScreenUpdating = True
```

```
Exit Sub  
End If
```

```
'Verify that user wants to overwrite existing workbooks  
If ynCreateWB = False Then  
Msg = "This run will overwrite data in the existing output workbooks." & vbCrLf & vbCrLf _  
& "Do you want to continue?"  
If MsgBox(Msg, vbYesNo + vbQuestion, "Confirm action") = vbNo Then  
Application.GoTo Reference:="ScenarioIndex"  
Application.ScreenUpdating = True  
Exit Sub  
End If  
End If
```

```
'Refresh Input Matrix A if requested by user  
Workbooks(ThisBook).Activate  
Application.GoTo Reference:="Refresh"  
If Selection.Value = True Then  
'Make sure workbook exists  
RefreshInputMatrix  
End If
```

```
'IMatrixSheet is among variable read from the Input Tables Sheet  
'These values are retained for the first year only.  
IMatrixDir = DefaultDirectory  
IMatrixBook = MainBook
```

```
'ynMultiYear indicates whether the user is performing an analysis for  
'multiple years (True) or a single year (False)  
If ynMultiYear Then  
y1 = iStartYear  
y2 = iEndYear  
e1 = 1  
e2 = NumEffects  
Else  
y1 = iYear  
y2 = iYear  
e1 = iStartEffect  
e2 = iEndEffect  
End If
```

```
*****  
**** WORKBOOKS ARE ONLY CREATED  
**** IF THE USER REQUESTS THEM  
*****
```

```
'Application.GoTo Reference:="CreateWB"  
'If Selection.Value = True Then
```

```
FirstYear = True  
For iYear = y1 To y2  
strCurYear = YearLabel(iYear)  
strPrevYear = PrevYearLabel(iYear)  
Call MakeBookNames  
For iEffect = e1 To e2  
EffectsSheet = EffectData(iEffect).InputPage  
Select Case EffectData(iEffect).EffectType
```

```
Case "MA"  
Call MAEffect  
Case "M"  
Call MEffect(EffectData(iEffect).EffectLabel)  
Case "V"  
Call VEffect  
Case "A"  
CType = EffectData(iEffect).EffectLabel  
Call AEffect  
End Select  
Next  
FirstYear = False  
Next  
'End If  
Application.Workbooks(MainBook).Activate  
Sheets(MainSheet).Activate  
Call Cleanup  
  
MsgBox "Done!"  
  
End Sub
```

Calculations - ModMAEffect

'This module contains the code for calculations pertaining to the "Type MA" effects.
'These are the effects, such as Cost Level, that use a multiplicative factor that is applied to all mail classes

Sub MAEffect()

'Creates the templates for all Cost Segments using Mail Classes and Cost Segments in Master

'Variable definitions

'R row pointer
'C column pointer
Dim R, C As Long

'I, J misc local counters
Dim I, J As Integer

'Pointer to last row that contains data
Dim LastRow As Long

'booBYOpen True if we had to open the Base Year workbook, False if it was already open
'booLoop True if a good value is found during processing, False if not
Dim booBYOpen, booLoop As Boolean

'Sql1 is used for misc SQL statements
Dim Sql1 As String

'CS is the Cost Segment counter
'strCS is a padded, text representation of the cost segment (e.g., "01")
Dim CS As Integer
Dim strCS As String

'Result is a return code from subroutine OpenWB
Dim Result As Integer

ThisBook = ActiveWorkbook.Name
ThisSheet = ActiveSheet.Name
'MainSheet = ThisSheet
'MainBook = ThisBook

EffectLabel = EffectData(iEffect).EffectName
StatusLabel = strCurYear & " " & EffectLabel

'This prevents the screen from scrolling all over the place during processing
Application.ScreenUpdating = False

'Open the FY workbook and make sure it has the Input Matrix in it.
'If the input matrix is not there, copy it from the Master workbook
Call OpenWB(EffectData(iEffect).WorkBookName, DefaultDirectory, Result)
Eflag = 0
On Error GoTo LinkTemplateError
Sheets(IMatrixSheet).Activate
On Error GoTo 0

```
If Eflag = 0 Then
    Application.DisplayAlerts = False
    Sheets(IMatrixSheet).Delete
    Application.DisplayAlerts = True
End If
Call GetInputMatrix

'Make sure we're on the main sheet

Workbooks(MainBook).Activate

'Find the last Components row that has data in it
Worksheets(ComponentSheet).Activate

Cells(65535, 5).Select
Selection.End(xlUp).Select
LastRow = Selection.Row

'Make sheet name for each Cost Segment
Workbooks(MainBook).Activate

For CS = 1 To MaxCS
    Workbooks(MainBook).Activate
    Worksheets(ComponentSheet).Activate
    R = 2
    booLoop = False
    Do Until R > LastRow
        Cells(R, 3).Select
        If Selection.Value = 1 Then
            Cells(R, 4).Select
            If Val(Selection.Value) = CS Then
                booLoop = True
                Exit Do
            End If
        End If
        R = R + 1
    Loop

'Skip components that are not on the list
If (Not booLoop) Then
    GoTo EndCSLoop
End If

strCS = Right("00" & Trim(Str(CS)), 2)
SheetName = "CS" & strCS

'See whether the specified workbook is already open
'==== Part 1
'1.1 Does the specified workbook exist?
'Open or create the specified workbook
Application.StatusBar = StatusLabel & ": Checking for file"
Call OpenWB(EffectData(iEffect).WorkBookName, DefaultDirectory, Result)
'Call OpenWB(FYWorkbook, FYDirectory, Result)

'1.2 Does it have a worksheet with the right name?
```

```
'make sure it's not already open
Application.StatusBar = StatusLabel & ": Checking for worksheet"
Eflag = 0
On Error GoTo LinkTemplateError
'This will generate Error 9 (subscript out of range) if the specified worksheet does not exist
On Error GoTo LinkTemplateError
Eflag = 0
Sheets(SheetName).Activate
'Delete the sheet if it already exists
If Eflag = 0 Then
    Application.DisplayAlerts = False
    Sheets(SheetName).Delete
    Application.DisplayAlerts = True
End If
Sheets.Add
ActiveSheet.Name = SheetName
J = Sheets.Count
Sheets(SheetName).Move After:=Sheets(J)
Range("A1:C1").Select
Selection.MergeCells = True
Selection.WrapText = True
Selection.Font.Color = vbBlue
Selection.Font.Bold = True
Selection.Font.Size = 12
'put in cost segment name and set row height
CostSegName = CSName(CS)
'Header change 8/4/04 to comply with standard report format
Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & _
    "Change Report" & Chr(10) & "Table " & iEffect + 1 & ". " & EffectLabel & Chr(10) & _
    "C/S " & CS & " " & CostSegName
'Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & _
    "Component Change Report" & Chr(10) & EffectLabel & Chr(10) & _
    "C/S " & CS & " " & CostSegName
'Selection.Value = EffectLabel & Chr(10) & "CS " & CS & " " & CostSegName & Chr(10) & "FY "
& strCurYear
Rows("1:1").RowHeight = 70
Cells(Row1 - 2, Col1 + 1).Select
Selection.Value = "Factor"

'Put the mail classes onto this sheet
On Error GoTo 0
CSSheetMode = " "
Call GetClasses

Columns("A:C").Select
Columns("A:C").EntireColumn.AutoFit

EndCSLoop:
Next

'Put all the components onto the worksheets

Workbooks(EffectData(iEffect).WorkBookName).Activate
'Workbooks(FYWorkbook).Activate
Call AddComps
```

ClassSubtotals

On Error GoTo 0
TemplateFormulas

EndTemplate:

Exit Sub

'This error routine is invoked if there are errors at critical points during the
'importing of the input matrix

LinkTemplateError:

Eflag = 1

Resume Next

End Sub

Sub TemplateFormulas()

'Places the Cost Level formulas on the Cost Segment worksheets

```
Dim R1, R2, C1, C2 As Long
Dim LastRow As Long
Dim VolumeFactor(1600) As Double
Dim Compon As Integer
Dim VFact As Double
Dim CSSheet As String
Dim LastCol(100) As Integer
Dim I As Integer
Dim Class As Integer
Dim LineFormula As String
Dim Result As Integer
Dim MatchComp As Variant
Dim FC As Long
```

```
'Puts the formulas onto the template(s)
'Read the inputs to find where things are
```

```
For I = 1 To 100
  LastCol(I) = Col1 + 2
Next
```

```
'Go to the Effects sheet and read into an array all the lines that have an entry in
'both the Components column (2) and the Cost Level Factor column (4),
'and which do NOT say "NOT USED" in the Title column (3)
Call OpenWB(InputWorkbook, InputDirectory, Result)
Workbooks(InputWorkbook).Activate
Sheets(EffectsSheet).Activate
Cells(65536, ECompon).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
```

```
*****
*****NOTE ADDED 8/4/04
***** NEED TO CHANGE HARD-CODED COLUMN VALUES TO USER-DEFINED VARIABLES
*** 9/1/04 DONE n. kay
*****
```

```
For R1 = 1 To LastRow
  Compon = 0
  VFact = 0
  Cells(R1, ECompon).Select
  If IsNumeric(Selection.Value) Then
    Compon = Selection.Value
    Cells(R1, ECompName).Select
    If Selection.Value <> "NOT USED" Then
      Cells(R1, EffectData(iEffect).InputCol).Select
      If IsNumeric(Selection.Value) Then
        VFact = Selection.Value
      End If
    End If
  End If
  If Compon > 0 And VFact <> 0 Then
    VolumeFactor(Compon) = VFact
  End If
```

Next

```
'Make the FY workbook the active book  
Workbooks(EffectData(iEffect).WorkBookName).Activate
```

```
'Go through the array of Cost Level Factors and put each one on its sheet  
For Compon = 1 To 1600
```

```
  If SegCompon(Compon) <> 0 Then  
    CSSheet = "CS" & Right("00" & CStr(SegCompon(Compon)), 2)  
    Workbooks(EffectData(iEffect).WorkBookName).Activate  
    Sheets(CSSheet).Activate  
    Application.StatusBar = strCurYear & ": Updating " & CSSheet  
  'Determine which row is the last one with data on this sheet  
  Cells(Row1 + 1, Col1).Select  
  Selection.End(xlDown).Select  
  LastRow = Selection.Row
```

```
'Find the column that has this component  
  Rows(ComponentRow).Select  
  Cells(ComponentRow, 1).Activate  
  Set MatchComp = Selection.Find(What:=Compon, _  
    After:=ActiveCell, _  
    LookIn:=xlFormulas, _  
    LookAt:=xlWhole, _  
    SearchOrder:=xlByRows, _  
    SearchDirection:=xlNext, _  
    MatchCase:=False, _  
    SearchFormat:=False)
```

```
If Not MatchComp Is Nothing Then  
  C1 = MatchComp.Column  
  Cells(Row1 - 2, C1).Select  
  'Cells(Row1 - 1, C1).Select  
  *****  
  *****added 4/15/04 to put a '--' if no CL factor *****  
  *****and to put a '--' for formula if no CL factor *****  
  *****and also to put in class subtotal formulas *****  
  *****
```

```
  If VolumeFactor(Compon) <> 0 Then  
    Selection.Value = VolumeFactor(Compon)  
  Else  
    Selection.Value = "--"  
    Selection.HorizontalAlignment = xlCenter  
  End If  
  For R1 = Row1 To LastRow  
    Cells(R1, Col1 + 2).Select  
    Class = Selection.Value  
    'Find the class in the Classes array  
    For FC = 1 To MaxClass  
      If Classes(FC).Component = Class Then  
        Exit For  
      End If  
    Next FC  
    If Class > 0 And Classes(FC).SubTot = False Then  
      Cells(R1, C1).Select
```

```
    If VolumeFactor(Compon) <> 0 Then
        LineFormula = "=InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet) & "* R"
    & Row1 - 2 & "C" & C1
        'LineFormula = "=InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet) & "* R"
    & Row1 - 1 & "C" & C1
    ElseIf VolumeFactor(Compon) = 0 Then
        LineFormula = "--"
        Selection.HorizontalAlignment = xlRight
    End If
    Selection.Formula = LineFormula
    Selection.NumberFormat = "#,##0"
End If
If Class > 0 And Classes(FC).SubTot = True Then
    Cells(R1, C1).Select
    Selection.Formula = ClassFormulas(Class)
    Selection.NumberFormat = "#,##0"
End If
Next
End If
End If
Next
```

```
ComponentSubtotals
MakeOutputMatrix
End Sub
```

B. Calculations - ModMEffect

'This module contains the code for calculations pertaining to the "Type M" effects.
'These are the effects, such as Non-volume workload and Add Workday that are multiplicative.

```
*****  
*****  
' Non-Volume Workload and Add Workday  
*****  
*****
```

Sub MEffect(NVAWMode As String)

'Control for the Non-Volume Workload and Add Workday routines

'The input argument, NVAWMode is:

- ' NV for Non-volume workload processing
- ' AW for Add Workday processing

'Variable definitions

'R row pointer
'C column pointer
Dim R, C As Long

'I, J misc local counters
Dim I, J As Integer

'Pointer to last row that contains data
Dim LastRow As Long

'booBYOpen True if we had to open the Base Year workbook, False if it was already open
'booLoop True if a good value is found during processing, False if not
Dim booBYOpen, booLoop As Boolean

'Sql1 is used for misc SQL statements
Dim Sql1 As String

'CS is the Cost Segment counter
'strCS is a padded, text representation of the cost segment (e.g., "01")
Dim CS As Integer
Dim strCS As String

'Result is a return code from subroutine OpenWB
Dim Result As Integer

'PrevBook is the workbook that precedes the one we're working with.
'It is the Mail Volume workbook if we're in NV, or the NV workbook if we're in AW
'CurBook is the workbook we are going to work with in this pass - either NV or AW
Dim PrevBook, CurBook As String

Call FindPreviousEffect
PrevBook = EffectData(InputEffect).WorkBookName
CurBook = EffectData(iEffect).WorkBookName
EffectLabel = EffectData(iEffect).EffectName

```
StatusLabel = strCurYear & " " & EffectLabel
```

```
'Open the previous workbook and make sure it has the Output Matrix in it.
```

```
'If the input matrix is not there, copy it from the Master workbook
```

```
Call OpenWB(PrevBook, FYDirectory, Result)
```

```
Eflag = 0
```

```
On Error GoTo NVAWError
```

```
Sheets("OutputMatrix").Activate
```

```
On Error GoTo 0
```

```
If Eflag > 0 Then
```

```
    MsgBox "Can't continue." & vbCrLf & vbCrLf & "Output Matrix is not present in " & PrevBook,  
    vbOKOnly + vbCritical, "Can't continue"
```

```
    Stop
```

```
End If
```

```
'If the current workbook already has an InputMatrix sheet, delete it.
```

```
Call OpenWB(CurBook, FYDirectory, Result)
```

```
Eflag = 0
```

```
On Error GoTo NVAWError
```

```
Sheets(FYIMatrix).Activate
```

```
On Error GoTo 0
```

```
If Eflag = 0 Then
```

```
    Sheets(FYIMatrix).Select
```

```
    Application.DisplayAlerts = False
```

```
    Sheets(FYIMatrix).Delete
```

```
    Application.DisplayAlerts = True
```

```
End If
```

```
'If the current workbook already has an OutputMatrix sheet, then delete that too.
```

```
Eflag = 0
```

```
On Error GoTo NVAWError
```

```
Sheets("OutputMatrix").Activate
```

```
On Error GoTo 0
```

```
If Eflag = 0 Then
```

```
    Application.DisplayAlerts = False
```

```
    Sheets("OutputMatrix").Delete
```

```
    Application.DisplayAlerts = True
```

```
End If
```

```
'Go back to the previous workbook to copy its OutputMatrix into the current workbook as Input  
Matrix
```

```
'Copy the previous output matrix to Input Matrix
```

```
Call OpenWB(PrevBook, FYDirectory, Result)
```

```
Sheets("OutputMatrix").Select
```

```
Sheets("OutputMatrix").Copy After:=Workbooks(CurBook).Sheets(1)
```

```
Workbooks(CurBook).Activate
```

```
Sheets("OutputMatrix").Select
```

```
Sheets("OutputMatrix").Name = FYIMatrix
```

```
Application.StatusBar = StatusLabel & ": Setting up file " & EffectData(iEffect).WorkBookName
```

```
'Insert the Cost Segment worksheets and Mail Class labels on each sheet
```

```
CSSheetMode = "ME"
```

```
Call AddCSSheet
```

```
'Place all the components on the worksheets  
Workbooks(EffectData(iEffect).WorkBookName).Activate  
Call AddComps
```

```
On Error GoTo 0  
Call NVAWFormulas(NVAWMode)
```

```
Exit Sub
```

```
NVAWError:  
Eflag = 1  
Resume Next
```

```
End Sub
```

Sub NVAWFormulas(NVAWMode)

'Places the Non-volume workload and Add Workday formulas on the Cost Segment worksheets

'The input argument, NVAWMode is:

- ' NV for Non-volume workload processing
- ' AW for Add Workday processing

Dim R1, R2, C1, C2 As Long

Dim LastRow As Long

Dim NVAWFactor(1600) As Double

Dim NVAWType(1600) As Long

Dim Compon, iC As Integer

Dim VFact As Double

Dim CSSheet As String

Dim LastCol(100) As Integer

Dim I As Integer

Dim Class As Integer

Dim LineFormula As String

Dim Result As Integer

'FactorType contains:

' -2 if there is no such component number

' -1 if there is no entry in the "Type" column

' 0 if the "Type" column is "A"

' Component Number if the "Type" column contains a component number

Dim FactorType As Long

'TypeCol is the column number for the "Type" field

'FactorCol is the column number for the "Factor" field

Dim TypeCol, FactorCol As Integer

'CurBook holds the name of the workbook we're dealing with

Dim CurBook As String

'Puts the formulas onto the template(s)

'Read the inputs to find where things are

For I = 1 To 100

 LastCol(I) = Col1 + 2

Next

For I = 1 To 1600

 NVAWType(I) = -2

Next

CurBook = EffectData(iEffect).WorkBookName

EffectLabel = EffectData(iEffect).EffectName

FactorCol = EffectData(iEffect).InputCol

TypeCol = FactorCol + 1

StatusLabel = strCurYear & " " & EffectLabel

'Go to the Effects sheet and read into an array all the lines that:

' - have an entry in the Components column (2)

' - do NOT say "NOT USED" in the Title column (3)

Call OpenWB(InputWorkbook, InputDirectory, Result)

Workbooks(InputWorkbook).Activate

Sheets(EffectsSheet).Activate

'NOTE 8/4/04 THIS NEXT SECTION IS Using hard-coded columns on Effects WS
'instead of user-defined
'for component numbers and component name - Change!!
'9/1/04 - NKay Changed to using variables for column numbers

```
Cells(65536, ECompon).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
'Initialize factor arrays to 0
For R1 = 1 To LastRow
    Compon = 0
    VFact = 0
    Cells(R1, ECompon).Select
    If IsNumeric(Selection.Value) Then
        Compon = Selection.Value
        Cells(R1, ECompName).Select
        FactorType = -1
        VFact = 0
        If Selection.Value <> "NOT USED" Then
            Cells(R1, FactorCol).Select
            If IsNumeric(Selection.Value) Then
                VFact = Selection.Value
                Cells(R1, TypeCol).Select
                If IsNumeric(Selection.Value) Then
                    FactorType = Selection.Value
                ElseIf UCase(Selection.Value) = "A" Then
                    FactorType = 0
                End If
            End If
        End If
    End If
    If Compon > 0 Then
        NVAWFactor(Compon) = VFact
        NVAWType(Compon) = FactorType
    End If
Next

'Make the NV or AW workbook the active book
Workbooks(CurBook).Activate

'Go through the array of Factors and put each one on its sheet
For iC = 1 To ComponentCount
    Compon = Components(iC).CompNumber

    If NVAWType(Compon) > -2 Then
        CSSheet = "CS" & Right("00" & CStr(SegCompon(Compon)), 2)
        Workbooks(CurBook).Activate
        Sheets(CSSheet).Activate
        Application.StatusBar = StatusLabel & ": Updating " & CSSheet
    'Determine which row is the last one with data on this sheet
        Cells(Row1 + 1, Col1).Select
        Selection.End(xlDown).Select
        LastRow = Selection.Row
```

```

'Find the column that has this component
Rows(ComponentRow).Select
Cells(ComponentRow, 1).Activate
Set MatchComp = Selection.Find(What:=Compon, _
    After:=ActiveCell, _
    LookIn:=xlFormulas, _
    LookAt:=xlWhole, _
    SearchOrder:=xlByRows, _
    SearchDirection:=xlNext, _
    MatchCase:=False, _
    SearchFormat:=False)
If Not MatchComp Is Nothing Then
    C1 = MatchComp.Column
    Cells(Row1 - 2, C1).Select
    *****
    '***Do NOT DISPLAY FACTOR Or FORMULAS          *****
    '***IF FACTOR = 0, but to put in a '--'          *****
    *****
    If NVAWFactor(Compon) <> 0 Then
        Selection.Value = NVAWFactor(Compon)
        Cells(Row1 - 1, C1).Select
        'Cells(Row1 - 1, C1).Select
        *****
        'NOTE ADDED 8/4/04 code here needs to be changed
        'LILLIAN DOES NOT WANT OPTION OF 'ALL' IF INPUT TABLE IS BLANK
        'SHE WANTS THE USER TO BE ASKED IF 'ALL' IS DESIRED DURING THE
        'PREPROCESSING CHECK OF INPUT TABLES
        'IN THIS CASE, FIRST ASK THE USER IF WANTS 'ALL' OR A SPECIFIC CLASS,
        'OR IF WANTS TO ABORT RUN
        *****

    If NVAWType(Compon) = 0 Then
        Selection.Value = "All"
    Else
        Selection.Value = NVAWType(Compon)
    End If
    Selection.HorizontalAlignment = xlCenter
Else
    Selection.Value = "--"
    Selection.HorizontalAlignment = xlCenter
    Cells(Row1 - 1, C1).Select
    Selection.Value = "--"
    Selection.HorizontalAlignment = xlCenter
End If

For R1 = Row1 To LastRow
    Cells(R1, Col1 + 2).Select
    Class = Selection.Value
    If Class > 0 Then
        Cells(R1, C1).Select

'Find the class in the Classes array
For FC = 1 To MaxClass
    If Classes(FC).Component = Class Then
        Exit For
    
```

```
End If
Next FC
If (NVAWFactor(Compon) <> 0 And (NVAWType(Compon) = 0 Or _
  NVAWType(Compon) = Class)) And Classes(FC).SubTot = False Then
  LineFormula = "=InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet) & "*"
R" & Row1 - 2 & "C" & C1
  Selection.Formula = LineFormula
Elseif Classes(FC).SubTot = True Then
  Selection.Formula = ClassFormulas(Class)
Else
  Selection.Value = "--"
  Selection.HorizontalAlignment = xlRight
End If
Selection.NumberFormat = "#,##0"
End If
Next
End If
End If
'End If
Next

Workbooks(CurBook).Activate
Piggybacks
ComponentSubtotals
MakeOutputMatrix
End Sub
```

```
*****
*****
' End Non-Volume Workload and Add Workday
*****
*****
```

C. Calculations - ModVEffect

'This module contains the code for calculations pertaining to the "Type V" effects.
'These are the effects, such as Mail Volume, which rely on a vector of multiplicative factors

'Sub VolAdjustMaster()

Sub VEffect()

'Controls the Volume Adjustment process flow

Dim C1, C2, R1, R2 As Long

Dim iV As Integer

Dim Result As Integer

Dim LastRow As Long

Dim PDir As String

'Open or activate the Volume Adjustment workbook

'Read the Volume Adjustment factors

'Open the VolAdj workbook

'Verify the existence of, or add tabs for all Cost Segments

'Format the tabs: add Mail Class labels

'Go through the Volume Adjustment column on the Effects sheet:

' If value = "V", apply Vol Adj factors to the component on that row

'Prepare the Output Matrix

'Open the Inputs workbook and go to the Volume Adjustment Factors sheet

*****NOTE ADDED 8/5/04 CHANGED TO USE EFFECTLABEL ARRAY INSTEAD OF
HARD-CODING

*****EFFECT NAME HERE

*****ALSO, THE COLUMNS IN VOLADJSHEET (THE NAME IS USER-DEFINED)

*****ARE HARD-CODED IN AND SHOULD BE USER-DEFINED

*****HOWEVER, THERE IS NO WAY AS OF NOW IF AN ADDITIONAL V EFFECT IS

ADDED

*****TO DEFINE THE WORKSHEET NAME CONTAINING ADJUSTMENT FACTORS.

EffectLabel = EffectData(iEffect).EffectName

'EffectLabel = "Mail Volume Change"

StatusLabel = strCurYear & " " & EffectLabel

Application.StatusBar = StatusLabel & ": Reading factors"

Call OpenWB(InputWorkbook, InputDirectory, Result)

Sheets(VolAdjSheet).Activate

'Read the Volume Adjustment Factors

'Find the last row of data

*****NOTE ADDED 8/4/04 - THIS SECTION OF CODE USES HARD-CODED

*****SHEET SET-UP, INSTEAD OF USER-DEFINED. REDO

**** 9/1/04 - NKay added new variables for these columns and removed hard-codes

Cells(65536, VClassCol).Select

Selection.End(xlUp).Select

LastRow = Selection.Row

'Start at the top of the first column looking for a cell with the word "CLASS".

C1 = VClassCol

C2 = VFactorCol

R1 = 1

Cells(R1, C1).Select

Do Until Selection.Value = "CLASS"

 R1 = R1 + 1

 Cells(R1, C1).Select

Loop

'The data begins in the next row, and continues to LastRow

iV = 0

For R2 = R1 + 1 To LastRow

 iV = iV + 1

 Cells(R2, C1).Select

 If IsNumeric(Selection.Value) Then

 VolAdjClass(iV) = Selection.Value

 Cells(R2, C2).Select

 If IsNumeric(Selection.Value) Then

 VolAdjFactor(iV) = Selection.Value

 End If

 End If

Next

'Open the VolAdj workbook

Application.StatusBar = strCurYear & ": " & EffectData(iEffect).EffectName & " - setting up workbook"

'Open the workbook

Call OpenWB(EffectData(iEffect).WorkBookName, DefaultDirectory, Result)

'Copy the previous output matrix to Input Matrix

'First see if we already have an Output Matrix

Windows(EffectData(iEffect).WorkBookName).Activate

Eflag = 0

On Error GoTo VEffErr

Sheets("OutputMatrix").Select

If Eflag = 0 Then

 Application.DisplayAlerts = False

 Sheets("OutputMatrix").Delete

 Application.DisplayAlerts = True

End If

Eflag = 0

On Error GoTo VEffErr

Sheets(FYIMatrix).Select

If Eflag = 0 Then

 Application.DisplayAlerts = False

 Sheets(FYIMatrix).Delete

 Application.DisplayAlerts = True

End If

'Go to the previous workbook and get the output matrix

Call FindPreviousEffect

'Use a different directory if ALL the following conditions are met:

'This run starts with the first effect

'This run does NOT start in the first year

'The user has specified a previous workbook (in PriorYearDir)

```
PDir = DefaultDirectory
If FirstYear And (iEffect = 1) Then
  If Len(Trim(PriorYearDir)) > 0 Then
    PDir = PriorYearDir
  End If
End If
Call OpenWB(EffectData(InputEffect).WorkBookName, PDir, Result)
Windows(EffectData(InputEffect).WorkBookName).Activate
Sheets("OutputMatrix").Copy After:=Workbooks(EffectData(iEffect).WorkBookName).Sheets(1)
'Go to the Mail Volume workbook and rename the just-copied sheet as "InputMatrix"
Workbooks(EffectData(iEffect).WorkBookName).Activate
Sheets("OutputMatrix").Select
```

```
Sheets("OutputMatrix").Name = FYIMatrix
'Insert the Cost Segment worksheets and Mail Class labels on each sheet
CSSheetMode = "VolAdj"
Call AddCSSheet
'Place all the components on the worksheets
```

```
Workbooks(EffectData(iEffect).WorkBookName).Activate
Call AddComps
```

```
'Add columns to the Cost Segment sheets for the effect
Application.StatusBar = strCurYear & ": " & EffectData(iEffect).EffectName & " - adding effect to
Cost Segments"
Call MakeAdjVolFormulas
Workbooks(EffectData(iEffect).WorkBookName).Activate
```

Piggybacks

```
'Prepare the output matrix
ComponentSubtotals
Call MakeOutputMatrix
CSSheetMode = ""
EffectLabel = ""
StatusLabel = ""
Exit Sub
```

```
VEffErr:
Eflag = 1
Resume Next
```

```
End Sub
```

Sub MakeAdjVolFormulas()

'This sub puts the formulas onto the template(s)

```
Dim R1, R2, C1, C2 As Long
Dim Rw As Long
Dim LastRow, LastRow2 As Long
Dim VolumeFactor(1600) As Double
Dim Compon As Integer
Dim VFact As Double
Dim CSSheet As String
Dim LastCol(100) As Integer
Dim I As Integer
Dim Class As Integer
Dim LineFormula As String
Dim LastC As Long
```

```
For I = 1 To 100
  LastCol(I) = Col1 + 3
Next
```

'Go to the Effects sheet and read into an array all the lines that have an entry in
'both the Components column (2) and have a "V", "I", or "IM" in the Volume Adjustment column
(5),

'and which do NOT say "NOT USED" in the Title column (3)

```
Workbooks(InputWorkbook).Activate
Sheets(EffectsSheet).Activate
Cells(65536, ECompon).Select 'Changed 9/1/04 to variable for column with component number
Selection.End(xlUp).Select
LastRow = Selection.Row
For R1 = 1 To LastRow
  'Return activation to the Inputs workbook inside the processing loop
```

```
*****
***** NOTE ADDED 8/4/04 THIS SECTION OF CODE USES HARD-CODED FORMATS FOR
INPUT
***** TABLE. NEED TO CHANGE TO USER-DEFINED FORMATTING
***** done 9/1/04 n kay
*****
```

```
Workbooks(InputWorkbook).Activate
Compon = 0
VFact = 0
Cells(R1, ECompon).Select
If IsNumeric(Selection.Value) Then
  Compon = Selection.Value
  Cells(R1, ECompName).Select
  If Selection.Value <> "NOT USED" Then
    Cells(R1, EffectData(iEffect).InputCol).Select
    If Selection.Value = "V" Then
      VFact = 1
    ElseIf Selection.Value = "I" Or Selection.Value = "IM" Then
      VFact = 2
    ElseIf Selection.Value = "M" Then
      VFact = 3
    End If
  End If
End If
```

```

If Compon > 0 And SegCompon(Compon) > 0 Then
  VolumeFactor(Compon) = VFact
'Make the FY Vol Adj workbook the active book
  CSSheet = "CS" & Right("00" & CStr(SegCompon(Compon)), 2)
  Workbooks(EffectData(iEffect).WorkBookName).Activate
  Sheets(CSSheet).Activate
  Application.StatusBar = strCurYear & ": Mail Volume: Updating " & CSSheet
'Determine which row is the last one with data on this sheet
  Cells(Row1 + 1, Col1).Select
  Selection.End(xlDown).Select
  LastRow2 = Selection.Row

'Find the column that has this component
  Rows(ComponentRow).Select
  Cells(ComponentRow, 1).Activate
  Set MatchComp = Selection.Find(What:=Compon, _
    After:=ActiveCell, _
    LookIn:=xlFormulas, _
    LookAt:=xlWhole, _
    SearchOrder:=xlByRows, _
    SearchDirection:=xlNext, _
    MatchCase:=False, _
    SearchFormat:=False)
If Not MatchComp Is Nothing Then
  C1 = MatchComp.Column
  If VFact = 1 Then
    '8/4/04 Changed to remove blank row - CHANGED BACK 8/5/04
    Cells(Row1 - 2, C1).Select
    'Cells(Row1 - 1, C1).Select
    Selection.Value = "Mail Vol"
    Selection.HorizontalAlignment = xlCenter
    *****
    ***** FORMULAS TO DO CLASS SUBTOTALS *****
    *****
  For Rw = Row1 To LastRow2
    Cells(Rw, Col1 + 2).Select
    Class = Selection.Value
    'Find the class in the Classes array
    For FC = 1 To MaxClass
      If Classes(FC).Component = Class Then
        Exit For
      End If
    Next FC
    If Class > 0 And Classes(FC).SubTot = False Then
      Cells(Rw, C1).Select
      LineFormula = "=InputMatrix!R" & (Compon + ROffset) & "C" & (Class + COffset) & "*"
      RC" & Col1 + 3
      Selection.Formula = LineFormula
      Selection.NumberFormat = "#,##0"
    ElseIf Class > 0 And Classes(FC).SubTot = True Then
      Cells(Rw, C1).Select
      Selection.Formula = ClassFormulas(Class)
      Selection.NumberFormat = "#,##0"
    End If
  End If

```

Next
End If

```
*****
***** VOLUME MIX ADJ *****
*****

If VFact = 3 Then
'This is a volume mix adjustment
  Cells(Row1 - 2, C1).Select
  Selection.Value = "Vol Mix"
  Selection.HorizontalAlignment = xlCenter
'Do the regular volume adjustment and put results on the Ratios page
'Set up the Ratios page first
  Worksheets("Ratios").Activate
'Find last column used in Ratios worksheet
  Cells(ComponentRow, 255).Select
  Selection.End(xlToLeft).Select
  LastC = Selection.Column + 1
  If LastC < ComponentOffset + 1 Then
    LastC = ComponentOffset + 1
  End If
  Columns(LastC).Select
  Selection.ColumnWidth = 12
  'Cells(Row1 - 4, LastC).Select
  Selection.WrapText = True
  '8/4/04 - Decrement all of Row1 - xx values by one - CHANGED BACK 8/5/04
  Cells(Row1 - 4, LastC).Select
  Selection.Value = "Unadjusted Mail Volume Effect for " & ComponentName(Compon)
  Selection.WrapText = True
  Selection.HorizontalAlignment = xlCenter
  Cells(Row1 - 3, LastC).Select
  Selection.Value = Compon
  Selection.HorizontalAlignment = xlCenter
  Cells(Row1 - 2, LastC).Select
  Selection.Value = "Unadj MV"
  Selection.HorizontalAlignment = xlCenter
  For Rw = Row1 To LastRow2 - 1
    Cells(Rw, Col1 + 2).Select
    Class = Selection.Value
    'Find the class in the Classes array
    For FC = 1 To MaxClass
      If Classes(FC).Component = Class Then
        Exit For
      End If
    Next FC
    If Class > 0 And Classes(FC).SubTot = False Then
      Cells(Rw, LastC).Select
      LineFormula = "=InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet) & "*"
RC" & Col1 + 3
      Selection.Formula = LineFormula
      Selection.NumberFormat = "#,##0"
    End If
  Next
  Cells(LastRow2, LastC).Select
  LineFormula = "=SUM(R" & Row1 & "C:R" & LastRow2 - 1 & "C)"
```

```

Selection.Formula = LineFormula
Selection.NumberFormat = "#,##0"
'Now go back to the cost segment page and to the mix adjustment
Worksheets(CSSheet).Select
For Rw = Row1 To LastRow2
    Cells(Rw, Col1 + 2).Select
    Class = Selection.Value
    For FC = 1 To MaxClass
        If Classes(FC).Component = Class Then
            Exit For
        End If
    Next FC
    If Class > 0 And Classes(FC).SubTot = False And Class < TotVVClass Then
        Cells(Rw, C1).Select
        LineFormula = "=if(Ratios!R" & LastRow2 & "C" & LastC & "<>0, (( InputMatrix!R" & _
            (Compon + ROffSet) & "C" & (TotVVClass + COffSet) & "/" (Ratios!R" & _
            LastRow2 & "C" & LastC & "+ InputMatrix!R" & (Compon + ROffSet) & "C" & _
            (TotVVClass + COffSet) & ")) * (Ratios!R" & Rw & _
            "C" & LastC & "+ InputMatrix!R" & (Compon + ROffSet) & "C" & (Class +
COffSet) & _
            ")) - InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet) & ", 0)"
        Selection.Formula = LineFormula
        Selection.NumberFormat = "#,##0"
    ElseIf Class > 0 And Classes(FC).SubTot = True Then
        Cells(Rw, C1).Select
        Selection.Formula = ClassFormulas(Class)
        Selection.NumberFormat = "#,##0"
    End If
Next
End If
*****
**** put '--' in column if no MV effect    ***
*****
If VFact = 0 Then
'No volume adjustment for this component
Cells(Row1 - 2, C1).Select
Selection.Value = "--"
Selection.HorizontalAlignment = xlCenter
For Rw = Row1 To LastRow2
    Cells(Rw, Col1 + 2).Select
    Class = Selection.Value
'Find the class in the Classes array
    For FC = 1 To MaxClass
        If Classes(FC).Component = Class Then
            Exit For
        End If
    Next FC
    If Class > 0 And Classes(FC).SubTot = True Then
        Cells(Rw, C1).Select
        Selection.Formula = ClassFormulas(Class)
        Selection.NumberFormat = "#,##0"
    ElseIf Class > 0 And Classes(FC).SubTot = False Then
        Cells(Rw, C1).Select
        Selection.Value = "--"
        Selection.HorizontalAlignment = xlRight
    End If

```

```
Next  
End If  
End If  
End If  
Next
```

```
'Ensure that the Mail Volume workbook is the active one when we're done here  
Workbooks(EffectData(iEffect).WorkBookName).Activate
```

```
End Sub
```

Calculations - ModAEffect

```
*****  
'Main control routine to do the cost reductions, other programs, or workyear mix.  
'First read in the input data  
'Next open or create the template workbook  
'Then distribute all three types of cost reductions, other programs, or workyear mix.  
'Then sum up distributions onto the cost segment pages in the template workbook  
'NOTE: Type 1 equates to COBOL model control string 7, where the program amount  
' is distributed to mail classes using the receiving component as the key  
' Type 2 equates to COBOL model control string 8, where the program amount  
' is first distributed to a list of components and then each component amount  
' is then distributed to mail classes using the receiving component as the key  
' Type 3 equates to COBOL model control string 15, where the program amount  
' is distributed to mail classes using a separate distribution key and then  
' added back to the receiving component.  
  
' For Type 2, if the list of components can be described using an existing  
' parent (subtotal) component then the parent component will appear in the distribution  
' key column of the input data. A routine will be called to read through the  
' component master list and get all of the relevant children of the parent  
' component in order to distribute the cost to those children.  
' If the list of components cannot be described using an existing  
' parent component then the component list will be included in the input data.  
,  
'1/17/05. Add new Type 4: Program cost is first spread among a group of components, then  
' a named distribution key is used to distribute the cost in all components.  
' Also, fix Type 2 so can have multiple effects going to the same distribution key  
' Currently, the program overwrites earlier effects going to the same key.  
*****
```

Sub AEffect()

```
Call CROPInit  
Call CROPWorkbook  
Call CROPDist  
Call CROPSumtoCS
```

End Sub

'CROPInit reads in the input data file for the cost reductions or other programs.

Sub CROPInit()

'Row counter
Dim R1 As Integer

'Holds last row on worksheet
Dim LastRow As Long

'Array index
Dim J As Integer

'Holds result from open workbook function
Dim Result As Integer

'Open the inputs data workbook and go to the appropriate worksheet
Result = 0
Call OpenWB(InputWorkbook, InputDirectory, Result)
Workbooks(InputWorkbook).Activate
Sheets(EffectData(iEffect).InputPage).Activate

'Find last row of cost reduction / other program definitions
Cells(65536, EffectData(iEffect).NameCol - 1).Select 'Use variable for column instead of hard-
code
Selection.End(xlUp).Select
LastRow = Selection.Row

'ReDim CROPDefs(LastRow)

'Initialize cost reduction / other program input data array
For J = 1 To MaxCROP
 CROPDefs(J).COAmount = 0
 CROPDefs(J).COCComp = 0
 CROPDefs(J).CODK = 0
 CROPDefs(J).COSubTot = 0
 CROPDefs(J).COSheet = ""
 CROPDefs(J).CORow = 0
Next J

'Read in cost reduction / other program input data definitions one by one
CROPAmt = EffectData(iEffect).AmtCol
CROPName = EffectData(iEffect).NameCol
CROPComp = EffectData(iEffect).CompCol
CROPCompName = EffectData(iEffect).CompNameCol
CROPDK = EffectData(iEffect).DKCol
CROPSubTot = EffectData(iEffect).SubTotCol
'CROPDKType = EffectData(iEffect).DKTypeCol
J = 0
For R1 = 2 To LastRow
 Cells(R1, CROPAmt).Select
 If Selection.Value <> 0 Then
 J = J + 1
 CROPDefs(J).COAmount = Selection.Value
 Cells(R1, CROPName).Select
 CROPDefs(J).COName = Selection.Value
 Cells(R1, CROPComp).Select

```
CROPDefs(J).COComp = Selection.Value
Cells(R1, CROPCompName).Select
CROPDefs(J).COCompName = Selection.Value
Cells(R1, CROPDK).Select
CROPDefs(J).CODK = Selection.Value
Cells(R1, CROPSubTot).Select
CROPDefs(J).COSubTot = Selection.Value
CROPDefs(J).CORow = R1
'Set type of effect. Type 2 = COBOL model control string 8
'We know this because there is no distribution key and
'no receiving component listed.

'Type 1 = COBOL model control string 7
'There is no distribution key component listed in the input data,
'and a receiving component is listed.

'Type 3 = COBOL model control string 15
'There is both a receiving component and a distribution key
'component listed in the input data.

'Type 4 = new added 1/17/05
'There is a distribution key component, but no receiving component because
'program will be spread among a list of components

'If CROPDefs(J).COComp = 0 And CROPDefs(J).CODK > 0 Then
'1/17/05 Changed definition of Type 2 to go along with new table format
If CROPDefs(J).COComp = 0 And CROPDefs(J).CODK = 0 Then
    CROPDefs(J).COType = 2
ElseIf CROPDefs(J).COComp > 0 And CROPDefs(J).CODK = 0 Then
    CROPDefs(J).COType = 1
ElseIf CROPDefs(J).COComp > 0 And CROPDefs(J).CODK > 0 Then
    CROPDefs(J).COType = 3
ElseIf CROPDefs(J).COComp = 0 And CROPDefs(J).CODK > 0 Then
    CROPDefs(J).COType = 4
End If
End If
Next R1

End Sub
```

'CROPWorkbook creates the appropriate template workbook and adds all needed
'worksheet pages to the template workbook.

Sub CROPWorkbook()

'WBName contains the name of the workbook template to be created
'IMName contains the name of the workbook whose output matrix should be copied
' to WBName's input matrix
'SheetName is used to build the sheet name to add
'EFlag will be set to 1 if the SheetName does not yet exist
'I, J are looping counters

Dim WBName As String
Dim IMName As String
Dim SheetName As String
Dim Eflag As Integer
Dim Result As Integer
Dim I, J As Integer

'Open the cost reduction / other program workbook and
'make sure it has the Input Matrix in it.
'If the input matrix is not there, copy it from the prior effect workbook

Call FindPreviousEffect
IMName = EffectData(InputEffect).WorkBookName
WBName = EffectData(iEffect).WorkBookName
EffectLabel = EffectData(iEffect).EffectName

StatusLabel = strCurYear & " " & EffectLabel

'Create or open the effect workbook in question
Call OpenWB(WBName, FYDirectory, Result)

Application.StatusBar = strCurYear & ": " & EffectData(iEffect).EffectName & " - setting up
workbook"

'If the current workbook already has an InputMatrix sheet, delete it.

Workbooks(WBName).Activate

Eflag = 0

On Error GoTo NoSheet

Sheets(FYIMatrix).Activate

On Error GoTo 0

If Eflag = 0 Then

 Sheets(FYIMatrix).Select

 Application.DisplayAlerts = False

 Sheets(FYIMatrix).Delete

 Application.DisplayAlerts = True

End If

'If the current workbook already has an OutputMatrix sheet, then delete that too.

Eflag = 0

On Error GoTo NoSheet

Sheets("OutputMatrix").Activate

On Error GoTo 0

If Eflag = 0 Then

 Application.DisplayAlerts = False

```
Sheets("OutputMatrix").Delete  
Application.DisplayAlerts = True  
End If
```

```
If IMatrixSheet = "OutputMatrix" Then  
    IMatrixSheet = "InputMatrix"  
End If
```

```
'Copy the prior effect output matrix to this workbook's Input Matrix  
Call OpenWB(IMName, FYDirectory, Result)  
'Windows(IMName).Activate  
Sheets("OutputMatrix").Select  
Sheets("OutputMatrix").Copy After:=Workbooks(WBName).Sheets(1)
```

```
'Go to the new effect workbook and rename the just-copied sheet as "InputMatrix"  
Workbooks(WBName).Activate  
Sheets("OutputMatrix").Select  
'TempSheet = IMatrixSheet  
Sheets("OutputMatrix").Name = FYIMatrix
```

```
'Insert the Cost Segment worksheets and Mail Class labels on each sheet  
CSSheetMode = " "  
Call AddCSSheet  
'Add Component names and numbers to cost segment pages on template  
Call AddComps  
'Initialize ClassFormulas array just in case it wasn't done in cost level  
ClassSubtotals
```

```
'Insert additional worksheets as needed -  
'    one to do all Type 1, and  
'    one for each Type 2, and  
'    one for each receiving component in Type 3.
```

```
'Add a sheet for each Type 2 or receiving component in Type 3  
'A new sheet for each Type 3 receiving component will be created only for  
'the first occurrence of this receiving component  
'Likewise, a new sheet to do all Type 1s will only be created for the  
'first occurrence of a Type 1 record in the input data.
```

```
'1/17/05 Added storage of sheet name in data structure, new Type 4, and changed Type2 sheet  
name
```

```
'    to accomodate more than one Type 2 with same distribution key
```

```
J = 1
```

```
Do While CROPDefs(J).COAmount <> 0  
    If CROPDefs(J).COType = 1 Then  
        SheetName = "Type1"  
        CROPDefs(J).COSheet = SheetName  
    ElseIf CROPDefs(J).COType = 2 Then  
        'Create a worksheet for each Type 2  
        SheetName = "Type2_Idx" & J  
        CROPDefs(J).COSheet = SheetName  
    ElseIf CROPDefs(J).COType = 3 Then  
        SheetName = "Type3_C" & CROPDefs(J).COComp  
        CROPDefs(J).COSheet = SheetName  
    ElseIf CROPDefs(J).COType = 4 Then
```

```
SheetName = "Type4_Idx" & J
CROPDefs(J).COSheet = SheetName
End If
Workbooks(WBName).Activate
On Error GoTo NoSheet
Eflag = 0
Sheets(SheetName).Activate
On Error GoTo 0
'Eflag > 0 means this workbook does not exist, because an error
'was generated when trying to activate the worksheet
'Add the worksheet to hold the distributions for the component getting a
'Type 3 effect.
'Delete the sheet if it already exists
If Eflag = 0 Then
    Application.DisplayAlerts = False
    Sheets(SheetName).Delete
    Application.DisplayAlerts = True
End If
'Sheet has been deleted or was not there. Add it now.
Sheets.Add
ActiveSheet.Name = SheetName
I = Sheets.Count
Sheets(SheetName).Move After:=Sheets(I)
Range("A1.C1").Select
Selection.MergeCells = True
If CROPDefs(J).COType = 1 Then
    Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & EffectLabel & Chr(10)
& _
    "Distributed and Added to the Same Component"
'Selection.Value = EffectLabel & Chr(10) _
& "Distributed and Added to the Same Component" & CS & Chr(10) & "FY" & strCurYear
Elseif CROPDefs(J).COType = 2 Then
'Create a worksheet for each Type 2
    Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & EffectLabel & Chr(10)
& _
    "Allocated to Listed Components"
'Selection.Value = EffectLabel & Chr(10) _
& "Allocated to Children of Subtotal Component " _
& CROPDefs(J).CODK & CS & Chr(10) & "FY" & strCurYear
Elseif CROPDefs(J).COType = 3 Then
    Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & EffectLabel & Chr(10)
& _
    "Distributed by RF Keys and Added to Component " & CROPDefs(J).COComp
'Selection.Value = EffectLabel & Chr(10) _
& "Distributed by RF Keys and Added to Component " _
& CROPDefs(J).COComp & CS & Chr(10) & "FY" & strCurYear
'Create a worksheet for each Type 2
Elseif CROPDefs(J).COType = 4 Then
    Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & EffectLabel & Chr(10)
& _
    "Allocated to Listed Components and Distributed on " & CROPDefs(J).CODK
End If
'Selection.Value = EffectLabel & Chr(10) & SheetName & CS & Chr(10) & strCurYear
Selection.WrapText = True
Selection.Font.Color = vbBlue
Selection.Font.Bold = True
```

```
Selection.Font.Size = 12
Call GetClasses
Cells(ComponentRow, ClassColumn - 1).Select
Selection.Value = "Component"
Cells(ComponentRow + 1, ClassColumn - 1).Select
Selection.Value = "Distribution Key"
Cells(ComponentRow + 2, ClassColumn - 1).Select
Selection.Value = "Amount"
Rows("1:1").RowHeight = 70
CSSheetMode = " "

    J = J + 1
Loop

Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub
```

'CROPDist distributes each cost reduction or other program amount.
'Read through the input data, and depending on the Type, the appropriate
'distribution routine will be called.

Sub CROPDist()

'Holds the children of a subtotal component.
Dim GetKids(100) As Long
'The number of the receiving component getting the effect.
Dim Comp As Long
'Counter to cycle through input data array.
Dim J As Integer
'The amount of this effect
Dim Amount As Double
'The name of the component getting the effect.
Dim strCompName As String
'The name of the effect.
Dim EffectName As String

'If CType = "CR" Then
' Application.StatusBar = strCurYear & ": Distributing Cost Reductions"
'Elseif CType = "OP" Then
' Application.StatusBar = strCurYear & ": Distributing Other Programs"
'Elseif CType = "WM" Then
' Application.StatusBar = strCurYear & ": Distributing Workyear Mix"
'End If

J = 1
Do While CROPDefs(J).COAmount <> 0
Amount = CROPDefs(J).COAmount
Comp = CROPDefs(J).COComp
strCompName = CROPDefs(J).COCompName
EffectName = CROPDefs(J).COName
If CROPDefs(J).COType = 1 Then
Call Type1(J)
Elseif CROPDefs(J).COType = 2 Then
Call Type2(J)
Elseif CROPDefs(J).COType = 3 Then
Call Type3(J)
Elseif CROPDefs(J).COType = 4 Then
Call Type4(J)
End If
J = J + 1
Loop

End Sub

'Type1 subroutine processes all the Type 1 entries, where the cost reduction or
'other program amount is distributed to mail classes using the receiving component
'as the distribution key

Sub Type1(Index)

'Component number getting effect, amount and name of effect.

Dim Comp As Long

Dim Amount As Double

Dim EffectName As String

'Last column and row of worksheet

Dim LastCol As Long

Dim LastRow As Long

'Counter for looping

Dim J As Long

'String to create formula that goes into worksheet

Dim LineFormula As String

'Get the input data needed to do a Type 1 effect

Amount = CROPDefs(Index).COAmount

Comp = CROPDefs(Index).CComp

EffectName = CROPDefs(Index).CName

'Go to the Type1 worksheet page and write the effect name, receiving component number,

'and the total effect amount on the first empty column

Sheets("Type1").Activate

'Find last row of mail classes on sheet

Cells(65536, ClassColumn).Select

Selection.End(xlUp).Select

LastRow = Selection.Row

'Find the last component on the cost segment page. If this is the first Type1 to go

'on the page then ComponentOffset contains the first column number to use.

Cells(ComponentRow, 255).Select

Selection.End(xlToLeft).Select

LastCol = Selection.Column

If LastCol < ComponentOffset Then

 LastCol = ComponentOffset - 1

End If

'Now write the information into the column.

Columns>LastCol + 1).Select

Selection.ColumnWidth = 12

'Cells(1, LastCol + 1).Select

Cells(ComponentRow - 1, LastCol + 1).Select

Selection.WrapText = True

Selection.Value = EffectName

Cells(ComponentRow, LastCol + 1).Select

Selection.Value = Comp

Cells(ComponentRow + 1, LastCol + 1).Select

Selection.Value = Comp

Cells(ComponentRow + 2, LastCol + 1).Select

Selection.Value = Amount

Selection.NumberFormat = "#,##0"

```
Cells>LastRow, LastCol + 1).Select  
LineFormula = "=R" & ComponentRow + 2 & "C"  
Selection.Formula = LineFormula  
Selection.NumberFormat = "#,##0"
```

'Go down the column and write the formula to distribute the effect amount based
'on the distribution of component costs in the input matrix.

```
For J = ClassOffset To LastRow - 1  
  Cells(J, ClassColumn).Select  
  Class = Selection.Value  
  'Find the class in the Classes array  
  For FC = 1 To MaxClass  
    If Classes(FC).Component = Class Then  
      Exit For  
    End If  
  Next FC  
  If Class > 0 And Classes(FC).SubTot = False Then  
    LineFormula = "=InputMatrix!R" & (Comp + ROffSet) & "C" & (Class + COffSet) & "/"  
InputMatrix!R" & (Comp + ROffSet) _  
    & "C" & (TotClass + COffSet) & " * R" & LastRow & "C" & LastCol + 1  
    Cells(J, LastCol + 1).Select  
    Selection.Formula = LineFormula  
    Selection.NumberFormat = "#,##0"  
  End If  
  If Class > 0 And Classes(FC).SubTot = True Then  
    Cells(J, LastCol + 1).Select  
    Selection.Formula = ClassFormulas(Class)  
    Selection.NumberFormat = "#,##0"  
  End If  
Next J  
  
End Sub
```

'Type3 subroutine processes all the Type 3 entries, where the cost reduction or
'other program amount is distributed to mail classes using the named distribution
'key

Sub Type3(Index)

'Receiving component, amount, name, and distribution key for type 3 effect.

Dim Comp As Long

Dim Amount As Double

Dim EffectName As String

Dim DK As Long

'Last column and row of worksheet

Dim LastCol As Long

Dim LastRow As Long

'Looping counter

Dim J As Long

'String to build formula that will go onto worksheet

Dim LineFormula As String

'Holds name for effects worksheet

Dim SheetName As String

'Get the input data needed to do a Type 3 effect

Amount = CROPDefs(Index).COAmount

Comp = CROPDefs(Index).COComp

EffectName = CROPDefs(Index).COName

DK = CROPDefs(Index).CODK

'Go to the Type3 worksheet page and write the effect name, receiving component number,
'distribution key component,

'and the total effect amount on the first empty column

SheetName = CROPDefs(Index).COSheet

Sheets(SheetName).Activate

'Find last row of mail classes on sheet

Cells(65536, ClassColumn).Select

Selection.End(xlUp).Activate

LastRow = Selection.Row

'Find the last component on the cost segment page. If this is the first Type1 to go
'on the page then ComponentOffset contains the first column number to use.

Cells(ComponentRow, 255).Select

Selection.End(xlToLeft).Select

LastCol = Selection.Column

If LastCol < ComponentOffset Then

 LastCol = ComponentOffset - 1

End If

'Now write the information into the column.

Columns>LastCol + 1).Select

Selection.ColumnWidth = 12

'Cells(1, LastCol + 1).Select

Cells(ComponentRow - 1, LastCol + 1).Select

Selection.WrapText = True

Selection.Value = EffectName

Cells(ComponentRow, LastCol + 1).Select

```
Selection.Value = Comp  
Cells(ComponentRow + 1, LastCol + 1).Select  
Selection.Value = DK  
Cells(ComponentRow + 2, LastCol + 1).Select  
Selection.Value = Amount  
Selection.NumberFormat = "#,##0"  
Cells(LastRow, LastCol + 1).Select  
LineFormula = "=R" & ComponentRow + 2 & "C"  
Selection.Formula = LineFormula  
Selection.NumberFormat = "#,##0"
```

'Go down the column and write the formula to distribute the effect amount based
'on the input matrix values for the distribution key component.

```
For J = ClassOffset To LastRow - 1  
  Cells(J, ClassColumn).Select  
  Class = Selection.Value  
  'Find the class in the Classes array  
  For FC = 1 To MaxClass  
    If Classes(FC).Component = Class Then  
      Exit For  
    End If  
  Next FC  
  If Class > 0 And Classes(FC).SubTot = False Then  
    LineFormula = "=InputMatrix!R" & (DK + ROffSet) & "C" & (Class + COffSet) &  
"/InputMatrix!R" & (DK + ROffSet) _  
    & "C" & (TotClass + COffSet) & " * R" & LastRow & "C" & LastCol + 1  
    Cells(J, LastCol + 1).Select  
    Selection.Formula = LineFormula  
    Selection.NumberFormat = "#,##0"  
  End If  
  If Class > 0 And Classes(FC).SubTot = True Then  
    Cells(J, LastCol + 1).Select  
    Selection.Formula = ClassFormulas(Class)  
    Selection.NumberFormat = "#,##0"  
  End If  
Next J  
  
End Sub
```

'Type2 subroutine processes all the Type 2 entries, where the cost reduction or
'other program amount is spread among a group of components and the amount for each is
distributed by that
'component

Sub Type2(Index)

'Receiving component, amount, name for this effect.
Dim Comp As Long
Dim Amount As Double
Dim EffectName As String

'Holds the subtotal component number, which is shorthand for the list of components
'to distribution the effect amount to.
Dim DK As Long

'If there is a subtotal compononet listed that means that the subtotal component is on the master
component list,
'which means that the distribution of the effect can be made by finding the 'children'
'on the master component list.
'If there is no subtotal component listed that means that the subtotal component is not on the
master component list,
'and the components to distribute the cost to is on the input data file.

Dim SubTot As Integer

'Last column and row on the worksheet
Dim LastCol As Long
Dim LastRow As Long

'Looping counters
Dim I, J, C1 As Long

'LineFormula is a string used to build the formula to go into the worksheet cell.
Dim LineFormula As String

'Holds the worksheet name for the Type 2 effect.
Dim SheetName As String

'An array to contain the components to distribute the
Dim CompList() As Long
ReDim CompList(Max2Comp) As Long

'Row for distribution key component in input data.
Dim IDK As Long

'input matrix total cost class amount for a component
Dim TotComp As Long

'String to hold worksheet name
Dim SaveBook As String

'Result of OpenWB function
Dim Result As Integer

'Current class amount in a component

Dim ThisAmount As Double

```
'Get the input data needed to do a Type 2 effect
Amount = CROPDefs(Index).COAmount
Comp = CROPDefs(Index).COComp
EffectName = CROPDefs(Index).COName
SubTot = CROPDefs(Index).COSubTot
```

```
CROPAmt = EffectData(iEffect).AmtCol
CROPName = EffectData(iEffect).NameCol
CROPComp = EffectData(iEffect).CompCol
CROPCompName = EffectData(iEffect).CompNameCol
CROPDK = EffectData(iEffect).DKCol
CROPSubTot = EffectData(iEffect).SubTotCol
'CROPDKType = EffectData(iEffect).DKTypeCol
```

```
'Get the component list to first distribute the cost to. Is the component is
'a subtotal component found on the master component list then FindChildren will
'get all of its children. Otherwise get the component list from the input data
'worksheet for this effect.
```

```
If SubTot > 0 Then
```

```
    Call FindChildren(SubTot, 2, CompList)
```

```
Else
```

```
    'Read in the components to spread among.
    'Row in workbook is equal to CROP data index + 1
    SaveBook = ActiveWorkbook.Name
    Call OpenWB(InputWorkbook, InputDirectory, Result)
    Sheets(EffectData(iEffect).InputPage).Activate
```

```
'Now find the last column on this row of the input data worksheet
Cells(CROPDefs(Index).CORow, 255).Select
Selection.End(xlToLeft).Select
LastCol = Selection.Column
```

```
'Check for data input errors and set LastCol when have valid data
Cells(CROPDefs(Index).CORow, LastCol).Select
Do While IsNumeric(Selection.Value) = False Or Selection.Value = 0
    LastCol = LastCol - 1
    Cells(CROPDefs(Index).CORow, LastCol).Select
Loop
```

```
'Fill in the CompList array with the components to distribute to
```

```
I = 1
```

```
For J = CROPSubTot + 1 To LastCol
```

```
    Cells(CROPDefs(Index).CORow, J).Select
```

```
    CompList(I) = Selection.Value
```

```
    I = I + 1
```

```
Next J
```

```
'0 element of CompList array holds count of number of elements in array
```

```
CompList(0) = I - 1
```

```
'Return to the effect template workbook
```

```
Call OpenWB(SaveBook, FYDirectory, Result)
```

```
End If
```

```
'Go to the Type2 worksheet page and write each receiving component in a column,
```

'plus the effect name, receiving component number, distribution key component,
'and the total effect amount on the last column

```
SheetName = CROPDefs(Index).COSheet  
Sheets(SheetName).Activate
```

```
'Find last row of mail classes on sheet  
Cells(65536, ClassColumn).Select  
Selection.End(xlUp).Activate  
LastRow = Selection.Row  
Cells(LastRow + 1, ClassColumn - 1).Select  
Selection.Value = "Input Matrix Total"
```

'Start in the first column for component information and continue to
'write the information into the column. Also write the total class amount in the
'input matrix for the component

```
C1 = ComponentOffset + 1  
TotComp = CompList(0) - 1  
I = 1
```

'First write the total effect amount in before the first component to distribute to.

```
LastCol = ComponentOffset  
Columns>LastCol).Select  
Selection.ColumnWidth = 12  
'Cells(1, LastCol).Select  
Cells(ComponentRow - 1, LastCol).Select  
Selection.WrapText = True  
Selection.Value = "Total Cost for " & EffectName
```

'Write program amount at the top of the column as documentation

```
Cells(ComponentRow + 2, LastCol).Select  
Selection.Value = Amount  
Selection.NumberFormat = "#,##0"  
Cells(ComponentRow, LastCol).Select  
Selection.Value = SubTot  
Selection.NumberFormat = "#,##0"  
Cells(LastRow, LastCol).Select  
LineFormula = "=R" & ComponentRow + 2 & "C"  
Selection.Formula = LineFormula  
'Cells(ComponentRow + 2, ClassColumn - 1).Select  
'Selection.Value = "Total Cost"
```

Do While C1 <= TotComp + ComponentOffset + 1

```
Columns(C1).Select  
Selection.ColumnWidth = 12  
Cells(ComponentRow - 1, C1).Select  
Selection.WrapText = True  
Selection.Value = EffectName  
Cells(ComponentRow, C1).Select  
Selection.Value = CompList(I)  
Cells(ComponentRow + 1, C1).Select  
Selection.Value = CompList(I)  
Cells(LastRow + 1, C1).Select  
LineFormula = "=InputMatrix!R" & (CompList(I) + ROffSet) & "C" & (TotClass + COffSet)  
Selection.Formula = LineFormula
```

```
I = I + 1  
C1 = C1 + 1  
Loop
```

```
'Sum up the total input matrix amounts for the components  
Cells(LastRow + 1, ComponentOffset).Select  
LineFormula = "=Sum(RC" & ComponentOffset + 1 & ":RC" & C1 - 1 & ")"  
Selection.Formula = LineFormula  
Selection.NumberFormat = "#,##0"
```

```
'Distribute the effect amount to individual component using the input matrix total cost  
'for each receiving component as a weight
```

```
'First write the input matrix amount for the component below the last class
```

```
LastCol = C1 - 1  
For C1 = ComponentOffset + 1 To LastCol  
    Cells(LastRow, C1).Select  
    LineFormula = "=R" & LastRow + 1 & "C" & C1 & "/R" & LastRow + 1 & "C" &  
ComponentOffset & _  
    " * R" & LastRow & "C" & ComponentOffset  
    Selection.Formula = LineFormula  
    Selection.NumberFormat = "#,##0"  
    Cells(ComponentRow + 2, C1).Select  
    LineFormula = "=R" & LastRow & "C"  
    Selection.Formula = LineFormula  
    Selection.NumberFormat = "#,##0"
```

```
Next C1
```

```
'Go down the rows in each component and across all the components in the columns  
'and write the formula to distribute the total effect amount based  
'on the distribution of total component costs in the input matrix.
```

```
For C1 = ComponentOffset + 1 To LastCol  
    Cells(LastRow + 1, C1).Select  
    'Get effect amount for this component  
    ThisAmount = Selection.Value  
    'If effect amount not equal to zero then distribute the effect amount to classes  
    If ThisAmount <> 0 Then  
        For J = ClassOffset To LastRow - 1  
            Cells(J, ClassColumn).Select  
            Class = Selection.Value  
            Cells(ComponentRow, C1).Select  
            Comp = Selection.Value  
            'Find the class in the Classes array  
            For FC = 1 To MaxClass  
                If Classes(FC).Component = Class Then  
                    Exit For  
                End If  
            Next FC  
            If Class > 0 And Classes(FC).SubTot = False Then  
                LineFormula = "=InputMatrix!R" & (Comp + ROffset) & "C" & (Class + COffset) &  
"/InputMatrix!R" & (Comp + ROffset) _  
                & "C" & (TotClass + COffset) & " * R" & LastRow & "C" & C1  
                Cells(J, C1).Select  
                Selection.Formula = LineFormula
```

```
        Selection.NumberFormat = "#,##0"  
    End If  
    If Class > 0 And Classes(FC).SubTot = True Then  
        Cells(J, C1).Select  
        Selection.Formula = ClassFormulas(Class)  
        Selection.NumberFormat = "#,##0"  
    End If  
Next J  
End If  
Next C1  
  
End Sub
```

'Type4 subroutine processes all the Type 4 entries, where the cost reduction or other program amount is spread among a group of components and distributed using the given distribution key

Sub Type4(Index)

'Receiving component, amount, name for this effect.

Dim Comp As Long

Dim Amount As Double

Dim EffectName As String

'Holds the subtotal component number, which is shorthand for the list of components to distribution the effect amount to.

Dim SubTot As Long

'Holds the distribution key component

Dim DK As Long

'If there is a subtotal component listed that means that the subtotal component is on the master component list,

'which means that the distribution of the effect can be made by finding the 'children' on the master component list.

'If there is no subtotal component listed that means that the subtotal component is not on the master component list,

'and the components to distribute the cost to is on the input data file.

'Last column and row on the worksheet

Dim LastCol As Long

Dim LastRow As Long

'Looping counters

Dim I, J, C1 As Long

'LineFormula is a string used to build the formula to go into the worksheet cell.

Dim LineFormula As String

'Holds the worksheet name for the Type 2 effect.

Dim SheetName As String

'An array to contain the components to distribute the

Dim CompList() As Long

ReDim CompList(Max2Comp) As Long

'input matrix total cost class amount for a component

Dim TotComp As Long

'String to hold worksheet name

Dim SaveBook As String

'Result of OpenWB function

Dim Result As Integer

'Current class amount in a component

Dim ThisAmount As Double

'Get the input data needed to do a Type 4 effect

```
Amount = CROPDefs(Index).COAmount  
EffectName = CROPDefs(Index).CName  
SubTot = CROPDefs(Index).COSubTot  
DK = CROPDefs(Index).CODK
```

```
CROPAmt = EffectData(iEffect).AmtCol  
CROPName = EffectData(iEffect).NameCol  
CROPDK = EffectData(iEffect).DKCol  
CROPSubTot = EffectData(iEffect).SubTotCol
```

```
'Get the component list to first distribute the cost to. Is the component is  
'a subtotal component found on the master component list then FindChildren will  
'get all of its children. Otherwise get the component list from the input data  
'worksheet for this effect.
```

```
If SubTot > 0 Then
```

```
    Call FindChildren(SubTot, 2, CompList)
```

```
Else
```

```
    'Read in the components to spread among.
```

```
    'Row in workbook is equal to CROP data index + 1
```

```
    SaveBook = ActiveWorkbook.Name
```

```
    Call OpenWB(InputWorkbook, InputDirectory, Result)
```

```
    Sheets(EffectData(iEffect).InputPage).Activate
```

```
'Now find the last column on this row of the input data worksheet
```

```
Cells(CROPDefs(Index).CORow, 255).Select
```

```
Selection.End(xlToLeft).Select
```

```
LastCol = Selection.Column
```

```
'Check for data input errors and set LastCol when have valid data
```

```
Cells(CROPDefs(Index).CORow, LastCol).Select
```

```
Do While IsNumeric(Selection.Value) = False Or Selection.Value = 0
```

```
    LastCol = LastCol - 1
```

```
    Cells(CROPDefs(Index).CORow, LastCol).Select
```

```
Loop
```

```
'Fill in the CompList array with the components to distribute to
```

```
I = 1
```

```
For J = CROPSubTot + 1 To LastCol
```

```
    Cells(CROPDefs(Index).CORow, J).Select
```

```
    CompList(I) = Selection.Value
```

```
    I = I + 1
```

```
Next J
```

```
'0 element of CompList array holds count of number of elements in array
```

```
CompList(0) = I - 1
```

```
'Return to the effect template workbook
```

```
Call OpenWB(SaveBook, FYDirectory, Result)
```

```
End If
```

```
'Go to the Type2 worksheet page and write each receiving component in a column,
```

```
'plus the effect name, receiving component number, distribution key component,
```

```
'and the total effect amount on the last column
```

```
SheetName = CROPDefs(Index).COSheet
```

```
Sheets(SheetName).Activate
```

```
'Find last row of mail classes on sheet
```

```
Cells(65536, ClassColumn).Select  
Selection.End(xlUp).Activate  
LastRow = Selection.Row  
Cells(LastRow + 1, ClassColumn - 1).Select  
Selection.Value = "Input Matrix Total"
```

'Start in the first column for component information and continue to
'write the information into the column. Also write the total class amount in the
'input matrix for the component

```
C1 = ComponentOffset + 1  
TotComp = CompList(0) - 1  
I = 1
```

'First write the total effect amount in before the first component to distribute to.

```
LastCol = ComponentOffset  
Columns>LastCol).Select  
Selection.ColumnWidth = 12  
'Cells(1, LastCol).Select  
Cells(ComponentRow - 1, LastCol).Select  
Selection.WrapText = True  
Selection.Value = "Total Cost for " & EffectName
```

'Write program amount at the top of the column as documentation

```
Cells(ComponentRow + 2, LastCol).Select  
Selection.Value = Amount  
Selection.NumberFormat = "#,##0"  
Cells(ComponentRow, LastCol).Select  
Selection.Value = SubTot  
Selection.NumberFormat = "#,##0"  
Cells>LastCol).Select  
LineFormula = "=R" & ComponentRow + 2 & "C"  
Selection.Formula = LineFormula  
'Cells(ComponentRow + 2, ClassColumn - 1).Select  
'Selection.Value = "Total Cost"
```

'Write Input Matrix amount for each component

```
Do While C1 <= TotComp + ComponentOffset + 1  
  Columns(C1).Select  
  Selection.ColumnWidth = 12  
  Cells(ComponentRow - 1, C1).Select  
  Selection.WrapText = True  
  Selection.Value = EffectName  
  Cells(ComponentRow, C1).Select  
  Selection.Value = CompList(I)  
  Cells(ComponentRow + 1, C1).Select  
  Selection.Value = DK  
  Cells>LastRow + 1, C1).Select  
  LineFormula = "=InputMatrix!R" & (CompList(I) + ROffSet) & "C" & (TotClass + COffSet)  
  Selection.Formula = LineFormula  
  I = I + 1  
  C1 = C1 + 1  
Loop
```

'Sum up the total input matrix amounts for the components
Cells>LastRow + 1, ComponentOffset).Select

```
LineFormula = "=Sum(RC" & ComponentOffset + 1 & ":RC" & C1 - 1 & ")"  
Selection.Formula = LineFormula  
Selection.NumberFormat = "#,##0"
```

'Distribute the effect amount to individual component using the input matrix total cost
'for each receiving component as a weight

'Calculate the program cost going to each component using InputMatrix amounts as the weight.

```
LastCol = C1 - 1  
For C1 = ComponentOffset + 1 To LastCol  
    Cells(LastRow, C1).Select  
    LineFormula = "=R" & LastRow + 1 & "C" & C1 & "/R" & LastRow + 1 & "C" &  
ComponentOffset & _  
    " * R" & LastRow & "C" & ComponentOffset  
    Selection.Formula = LineFormula  
    Selection.NumberFormat = "#,##0"  
    Cells(ComponentRow + 2, C1).Select  
    LineFormula = "=R" & LastRow & "C"  
    Selection.Formula = LineFormula  
    Selection.NumberFormat = "#,##0"  
Next C1
```

'Go down the rows in each component and across all the components in the columns
'and write the formula to distribute the total effect amount using the given distribution key.

```
For C1 = ComponentOffset + 1 To LastCol  
    Cells(LastRow + 1, C1).Select  
    'Get effect amount for this component  
    ThisAmount = Selection.Value  
    'If effect amount not equal to zero then distribute the effect amount to classes  
    If ThisAmount <> 0 Then  
        For J = ClassOffset To LastRow - 1  
            Cells(J, ClassColumn).Select  
            Class = Selection.Value  
            'Find the class in the Classes array  
            For FC = 1 To MaxClass  
                If Classes(FC).Component = Class Then  
                    Exit For  
                End If  
            Next FC  
            If Class > 0 And Classes(FC).SubTot = False Then  
                LineFormula = "=InputMatrix!R" & (DK + ROffSet) & "C" & (Class + COffSet) &  
"/InputMatrix!R" & (DK + ROffSet) _  
                & "C" & (TotClass + COffSet) & " * R" & LastRow & "C" & C1  
                Cells(J, C1).Select  
                Selection.Formula = LineFormula  
                Selection.NumberFormat = "#,##0"  
            End If  
            If Class > 0 And Classes(FC).SubTot = True Then  
                Cells(J, C1).Select  
                Selection.Formula = ClassFormulas(Class)  
                Selection.NumberFormat = "#,##0"  
            End If  
        Next  
    End If  
Next  
End If  
Next
```

End Sub

'CROPSumtoCS subroutine goes through the component list and finds the cost reduction /
'other program distributions for each component, and then writes a formula to
'sum up their amounts.

Sub CROPSumtoCS()

'I, J, C are various looping counters
Dim I, J, C As Long

'F is the index into the array containing the sheet name and column for the
'effect distributions for the receiving component in question.
Dim F As Long

'Used to build the formula to sum up the effects for the receiving component
Dim LineFormula As String

'Comp and CS are the component number and cost segment for a non-subtotal
'component that we will build a formula for to sum up all its effect distributions.
Dim Comp As Long
Dim CS As Long

'Last column and last row on a worksheet. Type3LastCol is the last column on the
'Type 3 worksheet for the receiving component.
Dim LastCol, LastRow, Type3LastCol As Long

'The current receiving component on a Type1 or Type2 worksheet
Dim ThisComp As Long

'Array to hold the sheet name and column number for effect distributions
Dim FindComp(10) As CompDist

'Used to build up and hold a worksheet name.
Dim SheetName As String

'Saves the Type 3 worksheet name if found for this receiving component
Dim SaveType3 As String

'The column on the cost segment page for the receiving component
Dim CompCol As Long

'Row counter for cycling through a cost segment page
Dim Row As Long

'strCS is used to build a cost segment page name
Dim strCS As String

'Various booleans - if a Type 3 page exists for this receiving component,
'if the receiving component has been found on the cost segment page,
'if at least one effect has already been added to the formula string for this
'receiving component

Dim booType3 As Boolean
Dim Found As Boolean
Dim DoneOne As Boolean
Dim FC As Long

'Cycle through the components array to pick up all non-subtotal components,
'then find all of the distributed effects for this component, put the worksheet
'name and column into an array. Will know if there is a Type 2 for this component
'by looking at the sheet names.

F = 1

'F will contain the number of entries for this receiving component in Type 1 and 2

For I = 1 To ComponentCount

'Use non-subtotal components only

If Components(I).CompSubTot = False And Components(I).CompCS <= LastCostCS Then

Comp = Components(I).CompNumber

CS = Components(I).CompCS

'Look for Type 1 distributions for this receiving component

On Error GoTo NoSheet

Eflag = 0

Sheets("Type1").Activate

On Error GoTo 0

If Eflag = 0 Then

Cells(ComponentRow, 255).Select

Selection.End(xlToLeft).Select

LastCol = Selection.Column

If LastCol >= ComponentOffset Then

For J = ComponentOffset To LastCol

Cells(ComponentRow, J).Select

ThisComp = Selection.Value

If ThisComp = Comp Then

FindComp(F).WS = "Type1"

FindComp(F).Col = J

F = F + 1

End If

Next J

End If

End If

'Next look in Type 2 distributions

'Look through the input data array for the Type 2s in order

'to find the distribution keys to make the sheet name

J = 1

Do While CROPDefs(J).COAmount <> 0

If CROPDefs(J).COType = 2 Or CROPDefs(J).COType = 4 Then

DK = CROPDefs(J).CODK

SheetName = CROPDefs(J).COSheet

Sheets(SheetName).Activate

Cells(ComponentRow, 255).Select

Selection.End(xlToLeft).Select

LastCol = Selection.Column

If LastCol >= ComponentOffset Then

For C = ComponentOffset To LastCol

Cells(ComponentRow, C).Select

ThisComp = Selection.Value

If ThisComp = Comp Then

FindComp(F).WS = SheetName

FindComp(F).Col = C

F = F + 1

```
        End If
    Next C
End If
End If
J = J + 1
Loop
```

'Now we know all of the cost reductions or other programs for
'this receiving component in Type 1, Type 2, and Type 4. We can easily
'find out if there is a Type 3 by checking to see if a sheet
'exists with the format SheetName = "Comp" & Comp

```
SheetName = "Type3_C" & Comp
On Error GoTo NoSheet
Eflag = 0
Sheets(SheetName).Activate
On Error GoTo 0
If Eflag > 0 Then
    booType3 = False
Else
    booType3 = True
    Cells(ComponentRow, 255).Select
    Selection.End(xlToLeft).Select
    Type3LastCol = Selection.Column
    SaveType3 = SheetName
End If
```

'Now we know where all of the receiving components are in all 3 types.
'Start building the formula to go onto the cost segment page.

```
'Open the cost segment sheet for the receiving component
strCS = Right("00" & Trim(Str(CS)), 2)
SheetName = "CS" & strCS
Sheets(SheetName).Activate
'Find last column on the page
Cells(ComponentRow, 255).Select
Selection.End(xlToLeft).Select
LastCol = Selection.Column
For J = ComponentOffset To LastCol
    Cells(ComponentRow, J).Select
    ThisComp = Selection.Value
    If ThisComp = Comp Then
        CompCol = J
        Found = True
        Exit For
    End If
Next J
```

'Found the component, so now build the formula, a line at a time.
'First get LastRow for the worksheet

```
Cells(65536, ClassColumn).Select
Selection.End(xlUp).Activate
LastRow = Selection.Row
```

'Build the formula, and then copy it down the row

```
'Effect found for this component
If booType3 = True Or F > 1 Then

    LineFormula = "=Sum("
    J = 1
    DoneOne = False
    'Remember that F - 1 = the number of entries in the FindComp array
    Do While J < F
        If J = 1 Then
            LineFormula = LineFormula & "" & FindComp(J).WS & "!RC" & FindComp(J).Col
            DoneOne = True
        Else
            LineFormula = LineFormula & "," & FindComp(J).WS & "!RC" & FindComp(J).Col
        End If
        J = J + 1
    Loop

    If booType3 = False Then
        LineFormula = LineFormula & ")"
    Else
        'now add the Type 3
        If DoneOne = True Then
            LineFormula = LineFormula & "," & SaveType3 & "!RC" & ComponentOffset & _
                ":RC" & Type3LastCol & ")"
        Else
            LineFormula = LineFormula & "" & SaveType3 & "!RC" & ComponentOffset & _
                ":RC" & Type3LastCol & ")"
        End If
    End If
Else
    LineFormula = "--"
End If
```

'Now copy the formula down the column for the receiving component

```
For Row = ClassOffset To LastRow
    Cells(Row, ClassColumn).Select
    ThisClass = Selection.Value
    'Find this class in the Classes array
    For FC = 1 To MaxClass
        If Classes(FC).Component = ThisClass Then
            Exit For
        End If
    Next FC
    If ThisClass > 0 And Classes(FC).SubTot = False And Found = True Then
        'If ThisClass > 0 And Classes(FC).SubTot = False Then
            Cells(Row, CompCol).Select
            Selection.Formula = LineFormula
            Selection.NumberFormat = "#,##0"
            Selection.HorizontalAlignment = xlRight
        End If
        If ThisClass > 0 And Classes(FC).SubTot = True And Found = True Then
            'If ThisClass > 0 And Classes(FC).SubTot = True Then
                Cells(Row, CompCol).Select
                Selection.Formula = ClassFormulas(ThisClass)
                Selection.NumberFormat = "#,##0"
```

```
        Selection.HorizontalAlignment = xlRight
    End If
    Next Row
'End If
    LineFormula = ""
    F = 1
End If

Next I

ComponentSubtotals
MakeOutputMatrix

Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub
```

D. Piggybacks - ModIndirect

```
*****  
*****  
***** START PIGGYBACKS SECTION - INDIRECT EFFECT COSTS *****  
*****  
*****
```

```
*****  
***          MAIN PIGGYBACK ROUTINE FOR CALCULATING INDIRECT COSTS          ***  
*****
```

'The Piggybacks subroutine calculates ratios for the piggybacks and puts formulas on cost
'segment pages to determine indirect costs.
'Ratios are calculated in the order that the distribution keys appear in the
'input workbook.
'EffectType is the name of the effect getting the indirect cost treatment.

'Set up to know first time go to GetRatios routine
'Public First As Boolean

Sub Piggybacks()

```
*****  
*** EDITED THIS MODULE JUN 11 2004 TO LOOK THROUGH THE DISTRIBUTION KEY  
ARRAY  
*** SELECT THE FINAL DISTRIBUTION KEY, AND THEN LOOK THROUGH ALL OTHER  
ADDENDS  
*** IN ALL OTHER KEYS AND ADD THEM TO THE BEFF AND AEFF PAGES. THIS  
RESOLVES  
*** TWO ISSUES - THE USE OF DK466 IN THE USPS VERSION, WHICH CONTAINS NON-  
LABOR  
*** COMPONENTS NOT FOUND IN THE ALL-LABOR KEY, AND THE PRC VERSION, WHICH  
DOES  
*** NOT INCLUDE ALL LABOR COMPONENTS FOUND IN OTHER DKS IN THE ALL-LABOR  
KEY  
*****
```

- '1. Set up worksheets to be used in generating ratios for indirect effects
' a) Create a worksheet with the before effect (input matrix) costs for
' all labor-related related components
' b) Create a worksheet with the after effect (cost segment page) costs for
' the same components. The formulas on this worksheet will automatically
' grab after effect costs for dependent components as they are calculated.
,
- '2. For each distribution key in the DKDefinitions file:
' a) Sum the before effect costs for the components comprising the key
' b) Sum the after effect costs for the components comprising the key
' c) Generate the formula for the ratio as (b) / (a)
' d) Write the ratio formula on on workbook page 'Ratios'
,
- '3. Calculate the indirect effect cost for all components using this DK
' by multiplying component costs from the input matrix by the Ratio
' and putting the formula to do this in the component column

4. Cycle through in order until all the distribution key definitions have been used.

'Assume a Ratios worksheet exists in the correct workbook

'Also assume the correct effect workbook will be open

'And columns exist on the cost segment pages for every component in the cost segment

'I, J are looping variables

'DKComponent contains the distribution key for this indirect component

'AddendCount contains the number of components comprising this distribution key

'CompList contains the component numbers for those components in this distribution key

Dim I, J As Integer

'Compon is the indirect component getting the indirect effect

Dim Compon As Integer

'DKComponent is the distribution key for the indirect component

Dim DKComponent As Integer

'AddendCount is the total number of components (addends) in the DK component

Dim AddendCount As Integer

'CompList will hold the list of components (addends) in the DK component

ReDim CompList(MaxDKComp) As Integer

'RatioDone is a flag to indicate a ratio has already been calculated for this DK

Dim RatioDone As Boolean

'Result of OpenWB function

Dim Result As Integer

'Just for testing purposes

Dim TestBook As String

'Total number of DK definitions in array.

Dim TotDefs As Long

'Error flag for finding a worksheet

Dim Eflag As Integer

'String holding worksheet name we want to go to

Dim SheetName As String

Dim strCS As String

'Row and column pointers

Dim C1, R1, C2 As Long

'Holds component numbers

Dim Comp, ThisComp As Long

'String holding component name

Dim strCompName As String

Dim NotFound As Boolean

Dim iTD As Long

Dim LastRow As Long
Dim LastCol As Integer

'First save the current workbook

TestBook = ActiveWorkbook.Name

'For testing only call DKDefinitions init. Should be called earlier in program
Call DKInit

'Fill in array containing the components requiring an indirect calculation for an effect
IndirectInit

Windows(TestBook).Activate

'Cycle through distribution key definitions and calculate ratios in order

'Find last row of distribution key definitions
'Read in distribution key definitions one by one

'Get the total number of definitions
TotDefs = DKDefinitions(0).DKNumber

***** Step I *****

'Set up worksheets that contain the before and after effect cost for all labor-
'related components. Use the last distribution key definition (All Salaries) as the
'definition of labor-related components
'Also add the distribution key components that comprise the DK 466. This is an
'unusual distribution key, as it is made up of other distribution keys rather than
'labor-related cost components.

'First create the worksheets

'The worksheet for the before effect costs from the Input Matrix:

SheetName = "BEFF"

On Error GoTo NoSheet

Eflag = 0

Sheets(SheetName).Activate

On Error GoTo 0

'Eflag > 0 means this worksheet does not exist, because an error
'was generated when trying to activate the worksheet

'Add the worksheet

If Eflag > 0 Then

 Sheets.Add

 ActiveSheet.Name = SheetName

 I = Sheets.Count

 Sheets(SheetName).Move After:=Sheets(I)

Else

 'Clear worksheet if it already exists

 Cells.Select

 Selection.ClearContents

```
End If
Range("A1.C1").Select
Selection.MergeCells = True
Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & EffectLabel & Chr(10) & _
    "Before Effects"
'Selection.Value = EffectLabel & Chr(10) & "Before Effects" & CS & Chr(10) & strCurYear
Selection.WrapText = True
Selection.Font.Color = vbBlue
Selection.Font.Bold = True
Selection.Font.Size = 12
Rows("1:1").RowHeight = 80
Cells(ComponentRow, ClassColumn - 1).Select
Selection.Value = "Component"
    'Put the classes on the worksheet
Call GetClasses
Columns("A:C").Select
Columns("A:C").EntireColumn.AutoFit

    'Put all of the labor component names and numbers on the sheet
AddendCount = DKDefinitions(TotDefs).DKAddends
For I = 1 To AddendCount
    C1 = I + (ComponentOffset - 1)
    Cells(ComponentRow, C1).Select
    Comp = DKDefinitions(TotDefs).DKComponents(I)
    Selection.Value = Comp
    strCompName = ComponentName(Comp)
    'Cells(1, C1).Select
    Cells(ComponentRow - 1, C1).Select
    Selection.Value = strCompName
    Selection.WrapText = True
    Columns(C1).Select
    Selection.ColumnWidth = 12
Next I

'Go through all the distribution key addends and add any not already
'in DKDefinitions(TotDefs).DKComponents

For J = 1 To TotDefs - 1
    If DKDefinitions(J).DKAddends > 0 Then
        For I = 1 To DKDefinitions(J).DKAddends
            Comp = DKDefinitions(J).DKComponents(I)
            NotFound = True
            For iTD = 1 To AddendCount
                If DKDefinitions(TotDefs).DKComponents(iTD) = Comp Then
                    NotFound = False
                    Exit For
                End If
            Next
            If NotFound = True Then
                AddendCount = AddendCount + 1
                C1 = AddendCount + (ComponentOffset - 1)
                Cells(ComponentRow, C1).Select
                Selection.Value = Comp
                'Cells(1, C1).Select
                Cells(ComponentRow - 1, C1).Select
                Selection.Value = ComponentName(Comp)
            End If
        Next
    End If
Next
```

```
        Selection.WrapText = True
        Columns(C1).Select
        Selection.ColumnWidth = 12
    End If
Next
End If
Next

'Now get the Input Matrix Costs to go onto the worksheet

'Find the last row with mail classes on the BeforeEffects page
Cells(65536, ClassColumn).Select
Selection.End(xlUp).Select
LastRow = Selection.Row

For C1 = ComponentOffset To ComponentOffset + AddendCount - 1
    Cells(ComponentRow, C1).Select
    Comp = Selection.Value
    Class = 0
    For R1 = ClassOffset To LastRow
        Cells(R1, ClassColumn).Select
        Class = Selection.Value
        'Find the class in the Classes array
        For FC = 1 To MaxClass
            If Classes(FC).Component = Class Then
                Exit For
            End If
        Next FC
        If Class > 0 And Classes(FC).SubTot = False Then
            LineFormula = "=InputMatrix!R" & (Comp + ROffSet) & "C" & (Class + COffSet)
            Cells(R1, C1).Select
            Selection.Formula = LineFormula
            Selection.NumberFormat = "#,##0"
        End If
        If Class > 0 And Classes(FC).SubTot = True Then
            Cells(R1, C1).Select
            Selection.Formula = ClassFormulas(Class)
            Selection.NumberFormat = "#,##0"
        End If
    Next R1
Next C1

'Now create a worksheet for after effect costs off the cost segment pages:
SheetName = "AEFF"
On Error GoTo NoSheet
Eflag = 0
Sheets(SheetName).Activate

On Error GoTo 0
'Eflag > 0 means this worksheet does not exist, because an error
'was generated when trying to activate the worksheet
'Add the worksheet
If Eflag > 0 Then
    Sheets.Add
    ActiveSheet.Name = SheetName
    I = Sheets.Count
```

```
Sheets(SheetName).Move After:=Sheets(l)
Else
'Clear worksheet if it already exists
Cells.Select
Selection.ClearContents
End If
Range("A1.C1").Select
Selection.MergeCells = True
Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & EffectLabel & Chr(10) & _
"After Effects"
'Selection.Value = EffectLabel & Chr(10) & "After Effects" & CS & Chr(10) & strCurYear
Selection.WrapText = True
Selection.Font.Color = vbBlue
Selection.Font.Bold = True
Selection.Font.Size = 12
Rows("1:1").RowHeight = 80
Cells(ComponentRow, ClassColumn - 1).Select
Selection.Value = "Component"
'Put the classes on the worksheet
Call GetClasses
Columns("A:C").Select
Columns("A:C").EntireColumn.AutoFit
'Put all of the labor component names and numbers on the sheet
AddendCount = DKDefinitions(TotDefs).DKAddends
For I = 1 To AddendCount
C1 = I + (ComponentOffset - 1)
Cells(ComponentRow, C1).Select
Comp = DKDefinitions(TotDefs).DKComponents(I)
Selection.Value = Comp
strCompName = ComponentName(Comp)
'Cells(1, C1).Select
Cells(ComponentRow - 1, C1).Select
Selection.Value = strCompName
Selection.WrapText = True
Columns(C1).Select
Selection.ColumnWidth = 12
Next I

'Go through all the distribution key addends and add any not already
'in DKDefinitions(TotDefs).DKComponents

For J = 1 To TotDefs - 1
If DKDefinitions(J).DKAddends > 0 Then
For I = 1 To DKDefinitions(J).DKAddends
Comp = DKDefinitions(J).DKComponents(I)
NotFound = True
For iTD = 1 To AddendCount
If DKDefinitions(TotDefs).DKComponents(iTD) = Comp Then
NotFound = False
Exit For
End If
Next
If NotFound = True Then
AddendCount = AddendCount + 1
C1 = AddendCount + (ComponentOffset - 1)
Cells(ComponentRow, C1).Select
```

```
        Selection.Value = Comp
        'Cells(1, C1).Select
        Cells(ComponentRow - 1, C1).Select
        Selection.Value = ComponentName(Comp)
        Selection.WrapText = True
        Columns(C1).Select
        Selection.ColumnWidth = 12
    End If
Next
End If
Next
```

'Put the after effects costs from the cost segment pages. These will be
'automatically be updated
'as each dependent component cost is calculated, as there are formulas in
'the cells to grab the costs off the cost segment pages

```
For C2 = ComponentOffset To ComponentOffset + AddendCount - 1
    Cells(ComponentRow, C2).Select
    Comp = Selection.Value
```

```
    'Get the cost segment page for this component
    CostSeg = SegCompon(Comp)
    strCS = Right("00" & Trim(Str(CostSeg)), 2)
    SheetName = "CS" & strCS
```

```
    'Find the component on the worksheet
    Sheets(SheetName).Activate
```

```
    'Find the last component on the cost segment page
    Cells(ComponentRow, 255).Select
    Selection.End(xlToLeft).Select
    LastCol = Selection.Column
```

```
    'Find the component on the cost segment page
```

```
    For C1 = ComponentOffset To LastCol
        Cells(ComponentRow, C1).Select
        ThisComp = Selection.Value
        If ThisComp = Comp Then
            Exit For
        End If
    Next C1
```

'Go back to 'After Effect' page and write the formula referencing the
'after effects costs on the cost segment page for this component

```
    Sheets("AEFF").Activate
    Class = 0
    For R1 = ClassOffset To LastRow
        Cells(R1, ClassColumn).Select
        Class = Selection.Value
        'Find the class in the Classes array
        For FC = 1 To MaxClass
            If Classes(FC).Component = Class Then
                Exit For
            End If
        Next FC
    Next R1
```

```
    End If
  Next FC
  If Class > 0 And Classes(FC).SubTot = False Then
    LineFormula = "=" & SheetName & "!R" & R1 & "C" & C1
    Cells(R1, C2).Select
    Selection.Formula = LineFormula
    Selection.NumberFormat = "#,##0"

    End If
  If Class > 0 And Classes(FC).SubTot = True Then
    Cells(R1, C2).Select
    Selection.Formula = ClassFormulas(Class)
    Selection.NumberFormat = "#,##0"
  End If
Next R1

Next C2

'The BeforeEffect and AfterEffect worksheets are now set up with formulas to
'grab the costs as they are calculated by the piggyback routine

***** Step II *****
'Cycle through the DKDefinitions array and calculate the ratios needed
'to do the indirect effect cost calculations
Application.StatusBar = strCurYear & " " & EffectLabel & ": Indirect"
Application.Calculation = xlManual
J = 1
Do While J < TotDefs + 1

  If DKDefinitions(J).DKNumber = 0 Then Exit Do
  DKComponent = DKDefinitions(J).DKNumber
  AddendCount = DKDefinitions(J).DKAddends
  For I = 1 To MaxDKComp
    CompList(I) = DKDefinitions(J).DKComponents(I)
  Next I

  'Cycle through the IndirectFactor array and find all the components using this
  'distribution key definition
  'First, initialize the RatioVector array
  For I = 1 To MaxClass
    RatioVector(I) = 0
  Next I

  'Calculate the ratios the first time this DK is seen in the DistributionKey array
  'All components getting an indirect effect with a specific DK
  'will all get EITHER a regular volume adjustment (Type 1)
  'OR a volume mix adjustment (Type 2).

  RatioDone = False
  For I = 1 To MaxComps
    Compon = I
    If RatioDone = False And IndirectFactor(I) > 0 And DistributionKey(I) = DKComponent Then
      IType = IndirectFactor(I)
      Call GetRatios(IType, CompList, AddendCount, DKComponent)
      Call WriteIndFormula(IType, Compon, DKComponent)
```

```
RatioDone = True
End If
If RatioDone = True And IndirectFactor(I) > 0 And DistributionKey(I) = DKComponent Then
    Call WriteIndFormula(IType, Compon, DKComponent)
End If
Next I
J = J + 1
Loop
Calculate
Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub
```

'The GetRatios subroutine is called by the main PIGGYBACK routine
'This routine takes the given distribution key and generates a formula to give the
'give the percentage change by mail class for before and after the effect for the
'component addends comprising the distribution key
'

'GetRatios is called with the following arguments:

' CompList is the array of components comprising the distribution key and
' that will get used to calculate the ratios
' AddendCount is the number of components in the CompList array
' DKComponent is the distribution key's component number

Sub GetRatios(IType, CompList, AddendCount, DKComponent)

'R1, C1, R2, C2 are row and column counters

Dim R1, C1 As Integer
Dim R2, C2 As Integer

'CostSeg, strCS and SheetName are used to find the cost segment sheet for a component
Dim CostSeg As Integer
Dim strCS As String
Dim SheetName As String

'Class is a class number label for a cost segment page row
Dim Class As Integer

'LastCol and LastRow are used to find the last column or last row used on a worksheet
Dim LastCol As Integer
Dim LastRow As Long

'Cost contains a cost value from the InputMatrix worksheet
Dim Cost As Double

'ThisAddend is a component that is added with other components to make a distribution key
Dim ThisAddend As Integer

'The last class to process for this effect
Dim LastClass As Integer

'Distribution key component number for All Salaries key
Dim AllSalaries As Long

'Looping variables
Dim I, J As Integer

'Column pointers on BeforeEffects and AfterEffects worksheets
Dim LastC1 As Long
Dim FirstC1 As Long

'Used to build formulas to go onto Ratios page
Dim LineFormula As String
Dim LineSumA As String
Dim LineSumB As String

Dim booSeq As Boolean

'Section I - No addends for the distribution key
'If AddendCount = 0 then then fill in Ratios page with the formula for
'ratios for the costs for the Distribution Key number itself,
'since there are no addend components

If AddendCount = 0 Then

 Sheets("BEFF").Activate

 'Find the last component on the BeforeEffects page
 Cells(ComponentRow, 255).Select
 Selection.End(xlToLeft).Select
 LastCol = Selection.Column

 'Find the component on the BeforeEffects sheet
 For C1 = ComponentOffset To LastCol
 Cells(ComponentRow, C1).Select
 Comp = Selection.Value
 If Comp = DKComponent Then
 Exit For
 End If
 Next C1

 'Go to Ratios page and write formula to calculate ratio
 Sheets("Ratios").Activate

 'Find the last row with mail classes on the Ratios page
 Cells(65536, ClassColumn).Select
 Selection.End(xlUp).Select
 LastRow = Selection.Row

 'Find the last component on the Ratios page
 Cells(ComponentRow, 255).Select
 Selection.End(xlToLeft).Select
 LastCol = Selection.Column

 'Nothing yet on the page
 If LastCol < ComponentOffset Then
 LastCol = ComponentOffset - 1
 End If

 'First write the title of the distribution key and the DK number
 'Cells(1, LastCol + 1).Select
 Cells(ComponentRow - 1, LastCol + 1).Select
 If IType = 1 Then
 Selection.Value = "Ratio for " & ComponentName(DKComponent)
 Selection.WrapText = True
 Cells(ComponentRow, LastCol + 1).Select
 Selection.Value = DKComponent
 Cells(ComponentRow + 1, LastCol + 1).Select
 Selection.Value = "Ratio"
 Columns(LastCol + 1).Select
 Selection.ColumnWidth = 12
 End If

```
'Build the formula to go into the Ratio page
LineFormula = "=if(BEFF!RC" & C1 & "<>0, SUM(AEFF!RC" & C1 & ")/SUM(" & _
    "BEFF!RC" & C1 & "), 0)"
```

```
'Put the formula on the Ratio page
For R1 = ClassOffset To LastRow
    Cells(R1, ClassColumn).Select
    Class = Selection.Value
    If (Class > 0 And Class <= MaxClass) Then
        Cells(R1, LastCol + 1).Select
        Selection.Formula = LineFormula
        Selection.NumberFormat = "#,##0.00000000"
    End If
Next R1
End If
```

'Section II. The distribution key is comprised of more than one component, which means that we must go through the addends (components) in the distribution key, and find all the before and after effects costs for each addend

'Length of formula limited to 1024 characters, so we'll shorten whenever we can using a range of columns.

```
If AddendCount > 0 Then
```

```
'Initialize the strings holding the formula. The strings will contain a long
'Sum formula for all the components(addends) comprising the DK
LineFormulaB = "Sum("
LineFormulaA = "Sum("
```

'Build Ratio Formula an addend at a time. GO to BEFF page, although the column location is the exact same on the AEFF page.

```
Sheets("BEFF").Activate
```

```
'Find the last component on the BeforeEffects page
Cells(ComponentRow, 255).Select
Selection.End(xlToLeft).Select
LastCol = Selection.Column
```

```
'Find each component comprising the DK on the BEFF worksheet
For R1 = 1 To AddendCount
    ThisAddend = CompList(R1)
    For C1 = ComponentOffset To LastCol
        Cells(ComponentRow, C1).Select
        Comp = Selection.Value
        If Comp = ThisAddend Then
            Exit For
        End If
    Next C1
```

'This booSeq is used to determine if the DK addends are in column order on the BEFF worksheet. If booSeq = True then they DK list follows the same order as the columns on the BEFF worksheet. This is useful because the SUM formula allows you to identify a range of cells. For example, we can

'write SUM(RC5:RC10), which will sum columns 5 through 10.
'LastC1 holds the last column looked at on the BEFF worksheet.
'FirstC1 holds the first column in the range of columns that are to be summed
'in order. In the example above, FirstC1 = 5.

```
If C1 = LastC1 + 1 And booSeq = False Then
    FirstC1 = LastC1
    booSeq = True
Elseif C1 = LastC1 + 1 And booSeq = True Then
    booSeq = True
Elseif C1 <> LastC1 + 1 Then
    booSeq = False
End If
```

'If booSeq is True then we are in a sequence of addends that are in order
'on the BEFF and AEFf worksheets. Keep going until we find the last addend in
'order on the BEFF and AEFf worksheets
'We are always one addend component behind when building the formula string
If booSeq = True And R1 < AddendCount Then
 LastC1 = C1

'At the end of a range of columns in order, so add the range to the
'formula string.
Elseif booSeq = False And R1 < AddendCount Then

```
'LastC1 = 0 for the first addend only
If LastC1 = 0 Then
    LastC1 = C1
    FirstC1 = 0
```

```
'The range is added to the formula string
Elseif LastC1 > 0 And FirstC1 > 0 Then
    LineFormulaB = LineFormulaB & "BEFF!RC" & FirstC1 & ":BEFF!RC" & LastC1 & ","
    LineFormulaA = LineFormulaA & "AEFF!RC" & FirstC1 & ":AEFF!RC" & LastC1 & ","
    LastC1 = C1
    FirstC1 = 0
```

```
'The component is in the DK, but is the column is not part of a range of
'columns. Add this single column to the formula string.
Elseif LastC1 > 0 And FirstC1 = 0 Then
    LineFormulaB = LineFormulaB & "BEFF!RC" & LastC1 & ","
    LineFormulaA = LineFormulaA & "AEFF!RC" & LastC1 & ","
    LastC1 = C1
    FirstC1 = 0
End If
```

'The last component (addend) of this distribution key, so finish off the string
'Have to add the prior column and the current column this time.
Elseif booSeq = False And R1 = AddendCount Then

```
'This column is not part of a range of columns, add it separately.
If FirstC1 = 0 Then
    LineFormulaB = LineFormulaB & "BEFF!RC" & LastC1 & ",BEFF!RC" & C1 & ")"
    LineFormulaA = LineFormulaA & "AEFF!RC" & LastC1 & ",AEFF!RC" & C1 & ")"
```

'The end of a range of columns to add to the formula string

```
Elseif FirstC1 > 0 Then
    LineFormulaB = LineFormulaB & "BEFF!RC" & FirstC1 & ":BEFF!RC" & LastC1 &
",BEFF!RC" & C1 & ")"
    LineFormulaA = LineFormulaA & "AEFF!RC" & FirstC1 & ":AEFF!RC" & LastC1 &
",AEFF!RC" & C1 & ")"
End If

'We are still in a range of columns, so go from FirstC1 to the last column
Elseif booSeq = True And R1 = AddendCount Then
    LineFormulaB = LineFormulaB & "BEFF!RC" & FirstC1 & ":BEFF!RC" & C1 & ")"
    LineFormulaA = LineFormulaA & "AEFF!RC" & FirstC1 & ":AEFF!RC" & C1 & ")"
End If

'Finish the formula. For Type 1, divide After Effects into Before Effects
If R1 = AddendCount And IType = 1 Then
    LineFormula = "=If(" & LineFormulaB & "<> 0," & LineFormulaA & "/" & LineFormulaB &
",0)"
'For Type 2, all we need is the sum of the BEFF and AEFF costs on the Ratio page.
Elseif R1 = AddendCount And IType = 2 Then
    LineFormula = "=" & LineFormulaA & "+" & LineFormulaB
End If

Next R1

'Go to Ratios page and write formula to calculate ratio
Sheets("Ratios").Activate

'Find the last row with mail classes on the Ratios page
Cells(65536, ClassColumn).Select
Selection.End(xlUp).Select
LastRow = Selection.Row

'Find the last component on the Ratios page
Cells(ComponentRow, 255).Select
Selection.End(xlToLeft).Select
LastCol = Selection.Column

If LastCol < ComponentOffset Then
    LastCol = ComponentOffset - 1
    'First = False
End If

'First write the title of the distribution key and the DK number
'Cells(1, LastCol + 1).Select
Cells(ComponentRow - 1, LastCol + 1).Select
If IType = 1 Then
    Selection.Value = "Ratio for " & ComponentName(DKComponent)
    Selection.WrapText = True
    Columns(LastCol + 1).Select
    Selection.ColumnWidth = 12
    Cells(ComponentRow, LastCol + 1).Select
    Selection.Value = DKComponent
    Cells(ComponentRow + 1, LastCol + 1).Select
    Selection.Value = "Ratio"
Elseif IType = 2 Then
```

```
Selection.Value = "After Effect Costs for " & ComponentName(DKComponent)
Selection.WrapText = True
Columns>LastCol + 1).Select
Selection.ColumnWidth = 12
Cells(ComponentRow, LastCol + 1).Select
Selection.Value = DKComponent
Cells(ComponentRow + 1, LastCol + 1).Select
Selection.Value = "Total"
End If

'Put the formula onto the Ratios page
For R1 = ClassOffset To LastRow
  Cells(R1, ClassColumn).Select
  Class = Selection.Value
  If (Class > 0 And Class <= MaxClass) Then
    Cells(R1, LastCol + 1).Select
    Selection.Formula = LineFormula
    If IType = 1 Then
      Selection.NumberFormat = "#,##0.00000000"
    ElseIf IType = 2 Then
      Selection.NumberFormat = "#,##0"
    End If
  End If
End For
Next R1
End If

End Sub
```

'WriteIndFormula is called by the main PIGGYBACK routine
'It writes the formula to do the indirect effect onto the cost segment page, using the
'ratio vector on the Ratios page for the distribution key for this indirect component.
,

'WriteIndFormula is called with the following arguments:
' Compon The Component getting the indirect effect
' DKComponent The Distribution Key associated with the before/after effects ratio

Sub WriteIndFormula(IType, Compon, DKComponent)

'LastRow, LastCol are used to find the last column and last row on a page
Dim LastRow As Long
Dim LastCol As Integer

'The column on the Ratio page containing the ratio we want
Dim RatioColumn As Integer

'The column on the cost segment page containing the indirect component
Dim CompColumn As Integer

'The following are used to go to the correct cost segment page
Dim CostSeg As Integer
Dim strCS As String
Dim SheetName As String

'Holds the distribution key number and the current class
Dim DK As Integer
Dim Class As Integer

'String where formula is built
Dim LineFormula As String

'Various counters
Dim I As Integer
Dim TotRow As Integer

Dim Usage As String

'First, find the column on the Ratios page for the distribution key ratios
Sheets("Ratios").Activate

'Find last column used on worksheet
Cells(ComponentRow, 255).Select
Selection.End(xlToLeft).Select
LastCol = Selection.Column
If LastCol < ComponentOffset Then
 LastCol = ComponentOffset
End If

RatioColumn = 0
For I = ComponentOffset To LastCol
 Cells(ComponentRow, I).Select
 DK = Selection.Value

```
Cells(ComponentRow + 1, I).Select
Usage = Selection.Value
If DK = DKComponent And (Usage = "Ratio" Or Usage = "Total") Then
    RatioColumn = I
    Exit For
End If
Next I
'Find the row for class TotClass
'Find last row used on worksheet
Cells(65536, 2).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
For I = ClassOffset To LastRow
    Cells(I, ClassColumn).Select
    Class = Selection.Value
    If Class = TotClass Then
        TotRow = I
    End If
Next I
```

```
'Find the cost segment page that needs formulas
CostSeg = SegCompon(Compon)
strCS = Right("00" & Trim(Str(CostSeg)), 2)
SheetName = "CS" & strCS
Sheets(SheetName).Activate
```

```
'Find last column used on worksheet
Cells(2, 255).Select
Cells(ComponentRow, 255).Select
Selection.End(xlToLeft).Select
LastCol = Selection.Column
If LastCol < ComponentOffset + 1 Then
    LastCol = ComponentOffset + 1
End If
```

```
'Find last row used on worksheet
Cells(65536, 2).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
```

```
'Now find the component on the cost segment page
CompColumn = 0
For I = ComponentOffset To LastCol
    Cells(ComponentRow, I).Select
    DK = Selection.Value
    If DK = Compon Then
        CompColumn = I
        Exit For
    End If
Next I
```

```
'Now write the formulas
```

```
'For Type 1, multiply Input Matrix costs (before effect) by the correct ratio
If IType = 1 Then
```

```

For R1 = ClassOffset To LastRow
    Cells(ComponentRow + 1, CompColumn).Select
    Selection.Value = "Indirect"
    Selection.HorizontalAlignment = xlCenter
    Cells(R1, ClassColumn).Select
    Class = Selection.Value
    'Find the class in the Classes array
    For FC = 1 To MaxClass
        If Classes(FC).Component = Class Then
            Exit For
        End If
    Next FC
    If Classes(FC).SubTot = False And Class > 0 Then
        Cells(R1, CompColumn).Select
        LineFormula = "=InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet) & "*"
        Ratios!R" & R1 & "C" & RatioColumn
        Selection.Formula = LineFormula
        Selection.NumberFormat = "#,##0"
    End If
    If Classes(FC).SubTot = True And Class > 0 Then
        Cells(R1, CompColumn).Select
        Selection.Formula = ClassFormulas(Class)
        Selection.NumberFormat = "#,##0"
    End If
Next R1
End If
    
```

'Type 2 is a indirect mix adjustment, which redistributes costs based on new volume mix without increasing total cost.
 'For Type 2, calculate a weight of total cost for the component on the input matrix divided by total costs for the DK components
 'Multiply class costs on Ratio page (Total BEFF+AEFF costs for DK components) by this weight.
 'Last, subtract out Input Matrix costs by class leaving only the change in cost by class.

```

If IType = 2 Then
    For R1 = ClassOffset To LastRow
        Cells(ComponentRow + 1, CompColumn).Select
        Selection.Value = "Indirect Mix"
        Selection.HorizontalAlignment = xlCenter
        Cells(R1, ClassColumn).Select
        Class = Selection.Value
        'Find the class in the Classes array
        For FC = 1 To MaxClass
            If Classes(FC).Component = Class Then
                Exit For
            End If
        Next FC
        If Classes(FC).SubTot = False And Class <= OtherClass And Class > 0 Then
            Cells(R1, CompColumn).Select
            If Class <= TotVVClass Then
                LineFormula = "=( InputMatrix!R" & (Compon + ROffSet) & "C" & (TotClass + COffSet)
                & " / Ratios!R" & TotRow & "C" & RatioColumn & " * Ratios!R" & R1 & "C" & RatioColumn & ") -
                InputMatrix!R" & (Compon + ROffSet) & "C" & (Class + COffSet)
                Selection.Formula = LineFormula
            End If
            If Class = OtherClass Then
                
```

```
LineFormula = "="&R[-1]C"  
Selection.Formula = LineFormula  
End If  
Selection.NumberFormat = "#,##0"  
End If  
If Classes(FC).SubTot = True And Class <= OtherClass And Class > 0 Then  
Cells(R1, CompColumn).Select  
Selection.Formula = ClassFormulas(Class)  
Selection.NumberFormat = "#,##0"  
End If  
Next R1  
End If  
  
End Sub
```

E. Subtotaling Routines - ModSubtotals

```
*****  
*****  
***** START SUBTOTALS MODULE *****  
*****  
*****
```

'This module contains the subroutine that puts in the class subtotal formulas into the
'cost segment workbooks.
'It should be run right before the output matrix is generated.

'This subroutine uses a public array that contains the row that a class is on on each
'cost segment page.

'This module will also put in the formulas for those components that are subtotals.
'In order for this to work the module assumes that all components already exist on the
'cost segment pages in the order that they appear in the Components list in the Master
'workbook.

```
*****  
*** PUBLIC CONSTANTS AND VARIABLES ***  
*****
```

```
'Public ClassRow(MaxClass) As Long  
'Public CSName(MaxCS) As String  
'Public ClassFormulas(MaxClass) As String
```

'Subroutine ClassSubtotals puts in the formulas to do the class subtotals
'It generates an array of subtotal formulas that should be put in when the
'rest of the formula goes down the column.
'Assumes that the correct workbook is already open and active.

Sub ClassSubtotals()

'I, J, K are looping counters
Dim I, J, K As Long

'strCS and SheetName are strings used for worksheet names
'LineFOrmula used to build cell formulas
Dim strCS As String
Dim SheetName As String
Dim LineFormula As String

'Last column on page, current column
Dim LastCol, ThisCol As Long

'Number of worksheets in the workbook
Dim NumSheets As Integer

'ParentClass is the parent class we generating a formula to subtotal its children
Dim ParentClass As Integer

'ChildClass is a child of the ParentClass
Dim ChildClass As Integer

'FirstSum is True if first child of the parent has not yet been added to formula
Dim FirstSum As Boolean

'The component for this column
Dim Comp As Long

'Cycle through the workbook.
'Find each cost segment page
'Go down the class column and find each subtotal class
'Build the formula to do that subtotal

'Go through the Classes array and find each Parent class

'Initialize the ClassRow array
InitClassRow

```
For J = 1 To MaxClass
  If Classes(J).SubTot = True Then
    ParentClass = Classes(J).Component
    'Build the formula for this Parent class
    'by finding all of the classes' children
    LineFormula = "=SUM("
    FirstSum = True
    For K = 1 To MaxClass
      If Classes(K).Parent = ParentClass Then
        ChildClass = Classes(K).Component
        ChildRow = ClassRow(ChildClass)
        If FirstSum = True Then
          LineFormula = LineFormula & "R" & ChildRow & "C"
          FirstSum = False
        Else
          LineFormula = LineFormula & ",R" & ChildRow & "C"
        End If
      End If
    End For
    If K = MaxClass Then
      LineFormula = LineFormula & ")"
    End If

    'Get the next child class for this parent class
    Next K
    ClassFormulas(ParentClass) = LineFormula
  End If
End For
```

'Get the next parent class
Next J

End Sub

'Initialize the ClassRow array with the row for each class.
'Run this before the FIRST time the subtotals will be run, i.e. at the
'end of the Cost Level effect, before the subtotals are put in, which
'is before the output matrix will be generated.

'Assume the workbook is already open and active.

Sub InitClassRow()

Dim I As Long
Dim LastRow As Long
Dim Class As Long

'Initialize the array to 0
For I = 1 To MaxClass
 ClassRow(I) = 0
Next I

'Pick any worksheet
Worksheets("CS01").Activate

'Find the last row on the worksheet
Cells(65536, ClassColumn).Select
Selection.End(xlUp).Select
LastRow = Selection.Row

For I = ClassOffset To LastRow
 Cells(I, ClassColumn).Select
 Class = Selection.Value
 If Class > 0 Then
 ClassRow(Class) = I
 End If
Next I

End Sub

'Subroutine ComponentSubtotals puts in the formulas in the subtotal component to
'sum up its child components

Sub ComponentSubtotals()

Dim I, J, K As Long
Dim LastCol, SubTotCol, ThisCol, C1 As Long
Dim LastRow As Long
Dim CS As Long
Dim strCS, SheetName As String
Dim LineFormula As String
Dim SubtotComp, ThisComp As Long
Dim CurComp As Long
Dim CompCol As Long
Dim Class As Long

'Go through the Components array and find each subtotal component.
'Find the subtotal component on the correct cost segment page.
'Then build a formula to add up the child components and put it
'in the subtotal component column.

'Initialize the ClassRow array
InitClassRow

Application.StatusBar = StatusLabel & ": Component Subtotals"
Application.Calculation = xlManual
For I = 1 To MaxCompon
 If Components(I).CompSubTot = True And Components(I).CompCS <= LastCostCS Then
 SubtotComp = Components(I).CompNumber
 CS = Components(I).CompCS

 LineFormula = "=SUM("
 FirstSum = True

 'Go to the worksheet for the cost segment of this subtotal component
 strCS = Right("00" & Trim(Str(CS)), 2)
 SheetName = "CS" & strCS
 Sheets(SheetName).Activate

 'First get the last column on the page
 Cells(ComponentRow, 255).Select
 Selection.End(xlToLeft).Select
 LastCol = Selection.Column

 'Make sure there are components on this cost segment page, and
 'then get the children
 If LastCol >= ComponentOffset Then

 'Find the parent component
 For C1 = ComponentOffset To LastCol
 Cells(ComponentRow, C1).Select
 CurComp = Selection.Value
 If CurComp = SubtotComp Then
 SubTotCol = C1
 End If

Next C1

```
Cells(ComponentRow + 1, SubTotCol).Select
If Components(ComponentIndex(SubtotComp)).CompTotOnly = True Then
    Selection.Formula = "Total CS" & strCS
    Selection.HorizontalAlignment = xlCenter
Else
    Selection.Value = "Subtotal"
    Selection.HorizontalAlignment = xlCenter
End If
```

'Go through Components array and find each child class

```
For J = 1 To MaxCompon
    If Components(J).CompParent = SubtotComp Then
        ChildComp = Components(J).CompNumber
        'Find the component on the page
        For K = ComponentOffset To LastCol
            Cells(ComponentRow, K).Select
            CurComp = Selection.Value
            If CurComp = ChildComp Then
                CompCol = K
            End If
        Next K
        If FirstSum = True Then
            LineFormula = LineFormula & "RC" & CompCol
            FirstSum = False
        Else
            LineFormula = LineFormula & ",RC" & CompCol
        End If
    End If
End If
```

```
End If
If J = MaxCompon Then
    LineFormula = LineFormula & ")"
End If
```

Next J

'Now write the formula in the parent component column

'The last row on the page is the row for the Total Cost class
LastRow = ClassRow(TotClass)

```
For R1 = ClassOffset To LastRow
    Cells(R1, ClassColumn).Select
    Class = Selection.Value
    If Class > 0 Then
        Cells(R1, SubTotCol).Select
        Selection.Formula = LineFormula
        Selection.NumberFormat = "#,##0"
        Selection.HorizontalAlignment = xlRight
    End If
Next R1
```

End If

End If

```
'Do the next subtotal component  
Next I  
Calculate  
End Sub
```

F. Error Checking Routines - ModErrorCheck

```
*****  
' This module contains all subroutines that perform error-checking of *  
' the Input files. This error-checking is done before the run is *  
' started so that any problems can be resolved prior to attempting to *  
' perform any calculations. *  
*****
```

Sub ErrorCheckDriver()

'Verifies that all files that are needed for the run are present.

Dim I, y1, y2, e1, e2 As Integer

Dim EffectsSheetFlag As Boolean

Dim Result As Integer

ErrorCount = 0

Application.StatusBar = "Error checking"

'Initialize loop limits

If ynMultiYear Then

 y1 = iStartYear

 y2 = iEndYear

 e1 = 1

 e2 = NumEffects

Else

 y1 = iYear

 y2 = iYear

 e1 = iStartEffect

 e2 = iEndEffect

End If

'Cycle through the years (outermost loop) worksheets (middle loop) and columns (innermost loop)

'Because of previous error-checking, we already know that the files exist.

'Sheet DistKeys will always be checked.

For I = y1 To y2

 Call OpenWB(InputTablesFile(I), DefaultDirectory, Result)

 EffectsSheetFlag = False

 For iEffect = e1 To e2

 If EffectData(iEffect).EffectType = "MA" Or EffectData(iEffect).EffectType = "V" Or

EffectData(iEffect).EffectType = "M" Then

 Call chkEffectList(EffectsSheetFlag)

 EffectsSheetFlag = True

 Elseif EffectData(iEffect).EffectType = "A" Then

 Call chkA

 Else

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Invalid Effect Type: " & EffectData(iEffect).EffectType

 End If

 Next

'Check the DK sheet

 Call chkDK

Next

End Sub

Sub chkEffectList(eFlg)

'Performs an error check for an effect on sheet EffectList
'The argument indicates whether this sheet has already been checked at least once for this year.
'If it has, then the component number column does NOT have to be checked again.
'The component title column is not checked by this routine.
'The "data" column is checked only for those rows that have a numeric (non-zero) entry in the
Component Number column

Dim C As Integer
Dim R, RLast As Long
Dim CS, CS1

'Make sure EffectsList worksheet exists in input table workbook
On Error GoTo NoSheet
Eflag = 0
Sheets(EffectData(iEffect).InputPage).Activate
On Error GoTo 0
If Eflag > 0 Then
 ErrorCount = ErrorCount + 1
 ReDim Preserve strErrorCheck(ErrorCount)
 strErrorCheck(ErrorCount) = "Worksheet " & EffectData(iEffect).InputPage & " does not exist in
 "
 " & ActiveWorkbook.Name
 Exit Sub
End If

'Find the last row of data (RLast)
C = ECompon
Sheets(EffectData(iEffect).InputPage).Activate
Cells(65536, C).Select
Selection.End(xlUp).Select
RLast = Selection.Row

'Cycle through them (allowing a title in the first row)
For R = 2 To RLast
 Cells(R, C).Select
 CS = Selection.Value
'Check for valid component number if this is the first check of this sheet
 If Not eFlg Then
 If Not ((IsNull(CS)) Or (CS = "") Or (IsComponent(CS))) Then
 ErrorCount = ErrorCount + 1
 ReDim Preserve strErrorCheck(ErrorCount)
 strErrorCheck(ErrorCount) = "Non-component entry in 'Component Number' column in " &
ActiveWorkbook.Name _
 & " on worksheet " & ActiveSheet.Name & " row " & R & ". "
 End If
 End If

'Check for valid data
If IsNumeric(CS) Then
 Select Case EffectData(iEffect).EffectType
 Case "MA"
 Cells(R, EffectData(iEffect).InputCol).Select
 CS1 = Selection.Value
 If Not ((IsNull(CS1)) Or (CS1 = "") Or (IsNumeric(CS1))) Then
 ErrorCount = ErrorCount + 1

```
ReDim Preserve strErrorCheck(ErrorCount)
strErrorCheck(ErrorCount) = "Non-numeric value in 'Cost Level Factor' column in " &
ActiveWorkbook.Name _
    & " on worksheet " & ActiveSheet.Name & " row " & R & "."
End If
Case "V"
Cells(R, EffectData(iEffect).InputCol).Select
CS1 = Selection.Value
If UCase(CS1) = "V" Or UCase(CS1) = "M" Then
Cells(R, EffectData(iEffect).InputCol + 1).Select
If Len(Trim(Selection.Value)) > 0 Then
ErrorCount = ErrorCount + 1
ReDim Preserve strErrorCheck(ErrorCount)
strErrorCheck(ErrorCount) = "Unexpected value in 'Volume Distribution Key' column
(expected this cell to be empty) in " _
    & ActiveWorkbook.Name & " on worksheet " & ActiveSheet.Name & " row
" & R & "."
End If
ElseIf UCase(CS1) = "I" Or UCase(CS1) = "IM" Then
Cells(R, EffectData(iEffect).InputCol + 1).Select 'look for a valid component
If Not IsNumeric(Selection.Value) Or Not IsComponent(Selection.Value) Then
ErrorCount = ErrorCount + 1
ReDim Preserve strErrorCheck(ErrorCount)
strErrorCheck(ErrorCount) = "Expected a valid component value in 'Volume Distribution
Key' column in " _
    & ActiveWorkbook.Name & " on worksheet " & ActiveSheet.Name & " row
" & R & "."
End If
End If
Case "M"
Cells(R, EffectData(iEffect).InputCol).Select
CS1 = Selection.Value
If IsNumeric(CS1) And Not IsEmpty(CS1) Then
Cells(R, EffectData(iEffect).InputCol + 1).Select
If Not (UCase(Selection.Value) = "A" Or IsClass(Selection.Value)) Then
ErrorCount = ErrorCount + 1
ReDim Preserve strErrorCheck(ErrorCount)
strErrorCheck(ErrorCount) = "Expected a class number or 'A' in " &
EffectData(iEffect).EffectLabel _
    & " 'Method' column in " _
    & ActiveWorkbook.Name & " on worksheet " & ActiveSheet.Name & " row
" & R & "."
End If
ElseIf UCase(CS1) = "I" Or UCase(CS1) = "IM" Then
Cells(R, EffectData(iEffect).InputCol + 1).Select
If Len(Trim(Selection.Value)) > 0 Then
ErrorCount = ErrorCount + 1
ReDim Preserve strErrorCheck(ErrorCount)
strErrorCheck(ErrorCount) = "Unexpected value in " & EffectData(iEffect).EffectLabel _
    & " 'Method' column (expected this cell to be empty) in " _
    & ActiveWorkbook.Name & " on worksheet " & ActiveSheet.Name & " row
" & R & "."
End If
End If
End Select
End If
```

Next

Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub

Private Sub chkA()

'Performs an error check on a Type A (CROP) input sheet

Dim Sht1 As String
Dim R As Long
Dim C As Integer

Sht1 = EffectData(iEffect).InputPage

'Make sure worksheet exists in input table workbook

On Error GoTo NoSheet

Eflag = 0

Sheets(Sht1).Activate

On Error GoTo 0

If Eflag > 0 Then

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Worksheet " & Sht1 & " does not exist in " _
 & ActiveWorkbook.Name

 Exit Sub

End If

'Start from Row 2 and keep going until a blank row is encountered.

R = 2

Do

 Cells(R, EffectData(iEffect).NameCol).Select

 If Len(Trim(Selection.Value)) = 0 Then

 Exit Sub

 End If

'Look for valid amount

 Cells(R, EffectData(iEffect).AmtCol).Select

 If Not IsNumeric(Selection.Value) Or IsEmpty(Selection.Value) Then 'Added check for empty
10/5

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Non-numeric value for program amount on sheet " & Sht1 & ",
row " & R

 End If

'Look for valid component

 Cells(R, EffectData(iEffect).CompCol).Select

 'Component Number may be blank if Distribution Key is not

 If (Not IsComponent(Selection.Value)) And (Len(Trim(Selection.Value)) > 0) Then

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Non-component entry in Component column on sheet " & Sht1 &
", row " & R

 Elseif IsNull(Selection.Value) Then

 Cells(R, EffectData(iEffect).DKCol).Select

 If Not IsComponent(Selection.Value) Then

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Component entry missing and no distribution key on sheet " &
Sht1 & ", row " & R

 End If

```
End If

'Look for valid DK designation
Cells(R, EffectData(iEffect).DKCol).Select
'Null is acceptable
If Len(Trim(Selection.Value)) > 0 Then
  If Not IsComponent(Selection.Value) Then
    ErrorCount = ErrorCount + 1
    ReDim Preserve strErrorCheck(ErrorCount)
    strErrorCheck(ErrorCount) = "Distribution Key error on sheet " & Sht1 & ", row " & R
  End If
End If

'Look for valid subtotal component
Cells(R, EffectData(iEffect).SubTotCol).Select
'Null is acceptable
If Len(Trim(Selection.Value)) > 0 Then
  If Not IsComponent(Selection.Value) Then
    ErrorCount = ErrorCount + 1
    ReDim Preserve strErrorCheck(ErrorCount)
    strErrorCheck(ErrorCount) = "Subtotal Component error on sheet " & Sht1 & ", row " & R
  End If
Else
  C = Selection.Column + 1
  Cells(R, C).Select
  Do Until Len(Trim(Selection.Value)) = 0
    If Not IsComponent(Selection.Value) Then
      ErrorCount = ErrorCount + 1
      ReDim Preserve strErrorCheck(ErrorCount)
      strErrorCheck(ErrorCount) = "Non-component entry in column " & C & " on sheet " & Sht1
    End If
    C = C + 1
    Cells(R, C).Select
  Loop
End If
R = R + 1
Loop

Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub
```

Private Sub chkDK()

'Checks the DK sheet for validity based on the following:

- '- If the "Used in RF" column is not marked "1", the line is not checked. All other checks are skipped
- '- If "DK Contents" is blank, then number of addends must be zero, and "Component Number" must be a valid component
- '- If "DK Contents" is not blank then "Number of addends" must be positive, and each addend must be a valid component
- '- The "Component Distributed" column is not checked.

Dim R As Long

Dim C, C1, C2 As Integer

'Make sure worksheet exists in input table workbook

On Error GoTo NoSheet

Eflag = 0

Sheets(DistKeysSheet).Activate

On Error GoTo 0

If Eflag > 0 Then

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Worksheet " & DistKeysSheet & " does not exist in " _
 & ActiveWorkbook.Name

 Exit Sub

End If

'Start from Row 2 and keep going until a blank row is encountered.

R = 2

Do

 Cells(R, DKNum).Select

 If Len(Trim(Selection.Value)) = 0 Then

 Exit Sub

 Elseif Not IsComponent(Selection.Value) Then 'Nancy added this next 4 lines to check DK num

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Distribution Key is not a valid component on sheet " &
DistKeysSheet & ", row " & R

 End If

 Cells(R, UsedInRF).Select 'Nancy added 10/5/04

 If Not IsNumeric(Selection.Value) Then

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Expected ""1"" or ""0"" in ""Used in RF"" column on sheet " &
DistKeysSheet & ", row " & R

 Else

 If Selection.Value = 1 Then

 Cells(R, AddendNum).Select 'Nancy changed to correct variable holding column name
10/5/04

 If Not IsNumeric(Selection.Value) Then

 ErrorCount = ErrorCount + 1

 ReDim Preserve strErrorCheck(ErrorCount)

 strErrorCheck(ErrorCount) = "Expected a number or ""0"" in ""Number of addends"" column
on sheet " & DistKeysSheet & ", row " & R

 Else

 If Selection.Value > 0 Then

```
C1 = Selection.Column + 1
C2 = C1 + Selection.Value - 1 'Nancy changed 10/5/04 to -1 so read correct column
For C = C1 To C2
  Cells(R, C).Select
  If Not IsComponent(Selection.Value) Then
    ErrorCount = ErrorCount + 1
    ReDim Preserve strErrorCheck(ErrorCount)
    strErrorCheck(ErrorCount) = "Non-component in Addends column " & C - C1 + 1 & " on
sheet " & DistKeysSheet & ", row " & R
  End If
Next
End If
End If
End If
R = R + 1
Loop

Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub
```

G. Initialization Routines - ModInit

Sub Init()

'Performs initializations that are needed throughout

Dim R1, I As Long

Dim Compon As Long

Dim ErrFlag As Integer

Dim LastRow As Long

Dim EMsg1, EMsg2 As String

'Following 2 lines specify which column and row headings should be started in

Col1 = 2

Col1 = 1

Row1 = 7

'INDIRECT, CROP, AND SUBTOTAL MODULES USE THESE

'VARIABLES THROUGHOUT CODE.

ComponentOffset = Col1 + 3

ClassColumn = Col1 + 2

'ComponentRow = Row1 - 2

ComponentRow = Row1 - 3

ClassOffset = Row1 + 2

ThisBook = ActiveWorkbook.Name

ThisSheet = ActiveSheet.Name

'Initialize MainBook and MainSheet only if they have not yet been initialized

If IsNull(MainBook) Or Len(MainBook) = 0 Then

 MainBook = ThisBook

End If

If IsNull(MainSheet) Or Len(MainSheet) = 0 Then

 MainSheet = ThisSheet

End If

'This prevents the screen from scrolling all over the place during processing

Application.ScreenUpdating = False

'Read in the names of the workbooks and worksheets that will be used.

ErrFlag = 0

On Error GoTo InitErr

'Read in the Base Year and Test Year values

EMsg1 = "trying to read the Base Year"

Application.GoTo Reference:="BaseYear"

BaseYear = Selection.Value + 1

EMsg1 = "trying to read the Test Year"

Application.GoTo Reference:="TestYear"

TestYear = Selection.Value

EMsg1 = "trying to read the Scenario"

Application.GoTo Reference:="ScenarioIndex"

ScenarioRow = Selection.Value + 1

```
'Read USPS/PRC analysis type
EMsg1 = "trying to read the USPS/PRC designator"
Application.GoTo Reference:="USPSRun"
USPSFlag = Selection.Value
If USPSFlag = 1 Then
    Application.GoTo Reference:="USPSSheet"
Else
    Application.GoTo Reference:="PRCSheet"
End If
ComponentSheet = ActiveSheet.Name
```

```
'Read InputMatrix Row and Column offsets
EMsg1 = "trying to read the Row Offset"
Application.GoTo Reference:="RowOffset"
ROffSet = Selection.Value
EMsg1 = "trying to read the Column Offset"
Application.GoTo Reference:="ColOffset"
COffSet = Selection.Value
```

```
ReDim RowHeaders(ROffSet - 1), ColHeaders(COffSet - 1) As String
```

```
'Read analysis type (Multi year or single year)
EMsg1 = "trying to read the Analysis Type (Multi-year or single year)"
'Application.GoTo Reference:="AnalysisType"
'This sets ynMultiYear to True if the first option is selected, False otherwise
'ynMultiYear = (Selection.Value = 1)
ynMultiYear = True
```

```
'Does user want program to create workbooks as necessary
EMsg1 = "trying to determine whether you want to create workbooks as needed"
Application.GoTo Reference:="CreateWB"
ynCreateWB = Selection.Value
```

```
'Read variables that are used to dimension arrays
EMsg1 = "trying to read Limits"
Application.GoTo Reference:="LimitsSheet"
R1 = Selection.Row
Cells(R1 + 1, 1).Select
MaxClass = Selection.Value
Cells(R1 + 2, 1).Select
MaxCS = Selection.Value
Cells(R1 + 3, 1).Select
MaxCompon = Selection.Value
Cells(R1 + 4, 1).Select
MaxEffects = Selection.Value
Cells(R1 + 5, 1).Select
MaxYears = Selection.Value
Cells(R1 + 6, 1).Select
InputClassStart = Selection.Value
Cells(R1 + 7, 1).Select
MaxDKComp = Selection.Value
Cells(R1 + 8, 1).Select
MaxComps = Selection.Value
Cells(R1 + 9, 1).Select
MaxDefs = Selection.Value
Cells(R1 + 10, 1).Select
```

```
TotVVClass = Selection.Value
Cells(R1 + 11, 1).Select
OtherClass = Selection.Value
Cells(R1 + 12, 1).Select
TotClass = Selection.Value
Cells(R1 + 13, 1).Select
MaxCROP = Selection.Value
Cells(R1 + 14, 1).Select
Max2Comp = Selection.Value
Cells(R1 + 15, 1).Select
LastCostCS = Selection.Value
'ReDim the arrays whose limits have just been read in
EMsg1 = "redimensioning arrays that are based on user-defined limits"
ReDim EffectData(MaxEffects)
ReDim AllWorkBookNames(MaxYears, MaxEffects)
ReDim VolAdjFactor(MaxClass)
ReDim VolAdjClass(MaxClass)
ReDim Classes(MaxClass)
ReDim Components(MaxCompon)
ReDim IndirectFactor(MaxComps)
ReDim DistributionKey(MaxComps)
ReDim RatioVector(MaxClass)
ReDim CROPDefs(MaxCROP)
ReDim ClassRow(MaxClass)
ReDim CSName(MaxCS)
ReDim ClassFormulas(MaxClass)
ReDim DKDefinitions(MaxDefs)

'Read file names and worksheet names
Call ReadFileNames(ErrFlag)
If ErrFlag > 0 Then
    EMsg1 = "reading file names you want to use"
    EMsg2 = "An error occurred during initialization while " & EMsg1
    MsgBox EMsg2, vbOKOnly + vbCritical, "Initialization error"
    Eflag = 1
    On Error GoTo 0
    Exit Sub
End If

'Read in a map of which components belong to which cost segments
EMsg1 = "building a map indicating which Components belong to which Cost Segments"
Sheets(ComponentSheet).Activate
Cells(65536, 1).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
For R1 = 1 To LastRow
    Cells(R1, 1).Select
    If IsNumeric(Selection.Value) Then
        Compon = Selection.Value
        If Compon > 0 And Compon < 10000 Then
            Cells(R1, 4).Select
            If IsNumeric(Selection.Value) Then
                If Selection.Value < 255 Then
                    SegCompon(Compon) = Selection.Value
                    Cells(R1, 2).Select
                    ComponentName(Compon) = Selection.Value
                End If
            End If
        End If
    End If
End For
```

```
End If  
End If  
End If  
End If  
Next
```

```
'Read all the component data into the Components array  
EMsg1 = "reading all the component data into the Components array"  
For I = 1 To 99999  
  ComponentIndex(I) = 0  
Next  
Sheets(ComponentSheet).Activate  
Cells(65536, 1).Select  
Selection.End(xlUp).Select  
LastRow = Selection.Row  
I = 0  
For R1 = 1 To LastRow  
  Cells(R1, 1).Select  
  If IsNumeric(Selection.Value) Then  
    I = I + 1  
    Components(I).CompNumber = Selection.Value  
    ComponentIndex(Components(I).CompNumber) = I  
    Cells(R1, 2).Select  
    Components(I).CompName = Selection.Value  
    Cells(R1, 3).Select  
    If IsNumeric(Selection.Value) Then  
      Components(I).CompLevel = Selection.Value  
    Else  
      Components(I).CompLevel = 0  
    End If  
    Cells(R1, 4).Select  
    Components(I).CompCS = 0  
    If IsNumeric(Selection.Value) Then  
      If Selection.Value < MaxCS Then  
        Components(I).CompCS = Selection.Value  
      End If  
    End If  
    Cells(R1, 5).Select  
    Components(I).CompParent = 0  
    If IsNumeric(Selection.Value) Then  
      Components(I).CompParent = Selection.Value  
    End If  
    Cells(R1, 6).Select  
    Components(I).CompSubTot = False  
    If Not IsNull(Selection.Value) Then  
      Components(I).CompSubTot = Selection.Value  
    End If  
    Cells(R1, 7).Select  
    Components(I).CompTotOnly = False  
    If Not IsNull(Selection.Value) Then  
      Components(I).CompTotOnly = Selection.Value  
    End If  
    Cells(R1, 8).Select  
    Components(I).CompHeader = False  
    If Not IsNull(Selection.Value) Then  
      Components(I).CompHeader = Selection.Value
```

```
End If
Cells(R1, 9).Select
Components(l).CompSort = Selection.Value
End If
Next
ComponentCount = l

'Clear the list of opened workbooks
For SCCount = 1 To 100
    SaveCloseNames(SCCount) = ""
Next
SCCount = 0
'See whether user wants to save/close workbooks and set the flag
EMsg1 = "setting the flag for whether you want to save/close workbooks"
Application.GoTo Reference:="SaveClose"
'coded this way so that default it True even if there is no value there
ynSaveClose = True
If Selection.Value = False Then
    ynSaveClose = False
End If

On Error GoTo 0
ClassArrayInit (ErrFlag)
If ErrFlag > 0 Then
    EMsg1 = "setting up the Class Array."
    EMsg2 = "An error occurred during initialization while " & EMsg1
    MsgBox EMsg2, vbOKOnly + vbCritical, "Initialization error"
    Eflag = 1
    On Error GoTo 0
    Exit Sub
End If
CSNameInit (ErrFlag)
If ErrFlag > 0 Then
    EMsg1 = "reading and setting up the Cost Segment Names."
    EMsg2 = "An error occurred during initialization while " & EMsg1
    MsgBox EMsg2, vbOKOnly + vbCritical, "Initialization error"
    Eflag = 1
    On Error GoTo 0
    Exit Sub
End If
InitializeWorkbookNames (ErrFlag)
If ErrFlag > 0 Then
    EMsg1 = "setting up the Workbook Names."
    EMsg2 = "An error occurred during initialization while " & EMsg1
    MsgBox EMsg2, vbOKOnly + vbCritical, "Initialization error"
    Eflag = 1
    On Error GoTo 0
    Exit Sub
End If

EExit:
On Error GoTo 0
Exit Sub

InitErr:
'Handles errors that occur during initialization
```

```
EMsg2 = "The following error occurred during initialization while " & EMsg1 & ":" & vbCrLf &  
vbCrLf _
```

```
    & "Error " & Err.Number & vbCrLf & vbCrLf & Err.Description
```

```
MsgBox EMsg2, vbOKOnly + vbCritical, "Initialization error"
```

```
Eflag = 1
```

```
Resume EExit
```

```
End Sub
```

Sub ReadFileNames(ErrFlag)

'Reads the names of all the input files and other data that is necessary to
'track the names of the files and where they get their data

'NOTE: Default directory MUST BE DONE FIRST.

Dim ColOffset, iEff, iC As Integer
Dim iR, Yr As Long
Dim Msg As String
Dim ELoc, ScenLoc As String

ELoc = "EffectsParameters"
ScenLoc = "Scenarios"

Workbooks(MainBook).Activate
Worksheets(ScenLoc).Activate

Cells(ScenarioRow, 2).Select
ScenName = Selection.Value
Cells(ScenarioRow, 3).Select
ScenDescription = Selection.Value
Cells(ScenarioRow, 4).Select
ScenWB = Selection.Value
Cells(ScenarioRow, 5).Select
DefaultDirectory = Selection.Value
If IsNull(DefaultDirectory) Or Len(Trim(DefaultDirectory)) = 0 Then
 Msg = "The default directory may not be left blank." & vbCrLf & vbCrLf _
 & "Fill in the default directory on the "" & Locations & "" sheet, then restart the analysis."
 MsgBox Msg, vbOKOnly + vbCritical, "Input Error"
 ErrFlag = 1
 Exit Sub

End If

'Put a backslash on the default directory if the user left it off

If Right(DefaultDirectory, 1) <> "\" Then
 DefaultDirectory = DefaultDirectory & "\"

End If

Cells(ScenarioRow, 6).Select
PriorYearDir = Selection.Value

If Len(PriorYearDir) > 0 And Right(PriorYearDir, 1) <> "\" Then
 PriorYearDir = PriorYearDir & "\"

End If

InputDirectory = DefaultDirectory

***** Input matrix A (200 x 1600) information

'Read Input matrix directory

Application.GoTo Reference:="InputMatrixADir"

IMatrixDir = DefaultDirectory

If Not IsNull(Selection.Value) And Len(Trim(Selection.Value)) > 0 Then
 IMatrixDir = Selection.Value

End If

If Right(IMatrixDir, 1) <> "\" Then

 IMatrixDir = IMatrixDir & "\"

End If

'Read Input Matrix workbook

Application.GoTo Reference:="InputMatrixAFile"

```
IMatrixBook = Selection.Value
'Read Input Matrix sheet name
Application.GoTo Reference:="InputMatrixASheet"
IMatrixSheet = Selection.Value
'Set Input Matrix data check flag to False
IMatrixCheck = False
***** end of Input matrix (200 x 1600) information
'Read name of worksheet in this book for starting worksheet
Application.GoTo Reference:="StartingInputMatrix"
IMatrixStart = Selection.Value

'Read in Distribution Keys worksheet name
Application.GoTo Reference:="RefDistKeysSheet"
DistKeysSheet = Selection.Value
'Read name of worksheet to be used in each effect workbook for the input matrix
Application.GoTo Reference:="IMatrixName"
FYIMatrix = Selection.Value
'Read name of worksheet with Volume Adjustment Factors
Application.GoTo Reference:="RefVolAdjSheet"
VolAdjSheet = Selection.Value

'Initialize variables containing column numbers for data on distribution key input sheet
Application.GoTo Reference:="UsedInRF"
UsedInRF = Selection.Value
Application.GoTo Reference:="DKCompNum"
DKNum = Selection.Value
Application.GoTo Reference:="DKAddends"
AddendNum = Selection.Value

'Initialize variables containing column numbers for data on effects worksheet
Application.GoTo Reference:="EffCompNum"
ECompon = Selection.Value
Application.GoTo Reference:="EffCompTitle"
ECompName = Selection.Value

'Initialize variables containing column number for volume adjustment factors worksheet
Application.GoTo Reference:="VClassNum"
VClassCol = Selection.Value
Application.GoTo Reference:="VVolFactor"
VFactorCol = Selection.Value

'Read in the effect-specific data
Application.GoTo Reference:="EffectsParmSheet"
'Find the last row of data
Selection.End(xlDown).Select
NumEffects = Selection.Row - 1
For iEff = 1 To NumEffects
    Cells(iEff + 1, 1).Select
    EffectData(iEff).EffectLabel = Selection.Value
    Cells(iEff + 1, 2).Select
    EffectData(iEff).EffectName = Selection.Value
    Cells(iEff + 1, 3).Select
    EffectData(iEff).EffectType = Selection.Value
    Cells(iEff + 1, 4).Select
    EffectData(iEff).InputMatrix = Selection.Value
    Cells(iEff + 1, 5).Select
```

```
EffectData(iEff).InputPage = Selection.Value  
Cells(iEff + 1, 6).Select  
EffectData(iEff).InputCol = Selection.Value  
Cells(iEff + 1, 7).Select  
EffectData(iEff).NameCol = Selection.Value  
Cells(iEff + 1, 8).Select  
EffectData(iEff).AmtCol = Selection.Value  
Cells(iEff + 1, 9).Select  
EffectData(iEff).CompCol = Selection.Value  
Cells(iEff + 1, 10).Select  
EffectData(iEff).CompNameCol = Selection.Value  
Cells(iEff + 1, 11).Select  
EffectData(iEff).DKCol = Selection.Value  
Cells(iEff + 1, 12).Select  
'EffectData(iEff).DKTypeCol = Selection.Value Removed 1/17/05 and replaced with SubTotCol  
EffectData(iEff).SubTotCol = Selection.Value  
Next
```

```
'Read Input Tables File names  
Application.GoTo Reference:="InputTablesSheet"  
iR = 2  
Do  
  Cells(iR, 1).Select  
  If Trim(Selection.Value) = "" Or IsNull(Selection.Value) Then Exit Do  
  YearLabel(iR - 1) = Selection.Value  
  Cells(iR, 2).Select  
  InputTablesFile(iR - 1) = Selection.Value  
  Cells(iR, 3).Select  
  PrevYearLabel(iR - 1) = Selection.Value  
  iR = iR + 1  
Loop
```

```
'Read starting and ending year or Year and starting and ending Effect  
If ynMultiYear Then  
  Application.GoTo Reference:="AnalysisStartYear"  
  iStartYear = Selection.Value  
  Application.GoTo Reference:="AnalysisEndYear"  
  iEndYear = Selection.Value  
  If iStartYear > iEndYear Then  
    Eflag = 1  
    MsgBox "Start Year and End Year are in the wrong order", vbOKOnly + vbCritical, "Action  
cancelled"  
  End If  
Else  
  Application.GoTo Reference:="AnalysisYear"  
  iYear = Selection.Value  
  Application.GoTo Reference:="StartEffect"  
  iStartEffect = Selection.Value  
  Application.GoTo Reference:="EndEffect"  
  iEndEffect = Selection.Value  
  If iStartEffect > iEndEffect Then  
    Eflag = 1  
    MsgBox "Start Effect and End Effect are in the wrong order", vbOKOnly + vbCritical, "Action  
cancelled"  
  End If  
End If
```

End Sub

Sub FileCheck()

'Verifies that all files that are needed for the run are present.

Dim I, J, K, y11, y12, e11, e12 As Integer

Dim F1 As String

Dim LocalEFlag As Integer

Dim EMsg, EMsgPart1, ETtl As String

LocalEFlag = 0

EMsgPart1 = "The following file is required for the analysis you requested, but was not found:" &
vbCrLf

EMsg = ""

'Initialize loop limits

If ynMultiYear Then

 y11 = iStartYear

 y12 = iEndYear

 e11 = 1

 e12 = NumEffects

Else

 y11 = iYear

 y12 = iYear

 e11 = iStartEffect

 e12 = iEndEffect

End If

'Check to see that the Input Tables files are present

For I = y11 To y12

 Eflag = 0

 F1 = DefaultDirectory & InputTablesFile(I)

 K = FreeFile

 On Error GoTo OpenFCErr

 Open F1 For Input As K

 Close K

 On Error GoTo 0

 If Eflag > 0 Then

 EMsg = EMsg & vbCrLf & F1

 End If

Next

'Check to see that the base year workbook exists, if this option is selected

'Added 10/5 by N. Kay

Workbooks(ThisBook).Activate

Application.GoTo Reference:="Refresh"

If Selection.Value = True Then

 F1 = IMatrixDir & IMatrixBook

 K = FreeFile

 On Error GoTo OpenFCErr

 Open F1 For Input As K

 Close K

 On Error GoTo 0

 If Eflag > 0 Then

 EMsg = EMsg & vbCrLf & F1

 End If

End If

'If user has NOT checked "Create workbooks as needed",
'make sure all the workbooks that are needed are present.

```
If Not ynCreateWB Then
  For I = yI1 To yI2
    For J = eI1 To eI2
      Eflag = 0
      F1 = DefaultDirectory & AllWorkbookNames(I, J)
      K = FreeFile
      On Error GoTo OpenFCErr
      Open F1 For Input As K
      Close K
      On Error GoTo 0
      If Eflag > 0 Then
        EMsg = EMsg & vbCrLf & F1
      End If
    Next
  Next
End If
'If any errors were found (missing files that are needed)
'then display an error message
'10/5 NK moved this out of above loop so it will execute even if user checks "Create Workbooks
as Needed"
If LocalEFlag > 0 Then
  Eflag = 1
  ETtl = "Input File Missing"
  If LocalEFlag > 1 Then
    EMsgPart1 = "The following files are required for the analysis you requested, but were not
found:" & vbCrLf
    ETtl = "Input Files Missing"
  End If
  EMsg = EMsgPart1 & EMsg & vbCrLf & vbCrLf & "The analysis cannot be run. Check your
input specifications and re-try."
  MsgBox EMsg, vbOKOnly + vbCritical, ETtl
End If

Exit Sub

OpenFCErr:
Eflag = 1
LocalEFlag = LocalEFlag + 1
Resume Next

End Sub

' DKInit reads in Distribution Key input table
' Do this once for each rollforward model run
```

Sub DKInit()

```
Dim MComp As Integer
Dim I, J As Integer
Dim Result As Integer
Dim LastRow As Long
Dim AddendOffset As Integer
```

```
'Open the inputs data workbook
Result = 0
Call OpenWB(InputWorkbook, InputDirectory, Result)
Workbooks(InputWorkbook).Activate
Sheets(DistKeysSheet).Activate
```

```
'Find last row of distribution key definitions
Cells(65536, 1).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
```

```
'Read in distribution key definitions one by one
```

```
J = 1
For R1 = 2 To LastRow
    DKDefinitions(J).DKOrder = 0 'jun 21 - column not used, set to 0
    Cells(R1, DKNum).Select
    DKDefinitions(J).DKNumber = Selection.Value
    Cells(R1, AddendNum).Select
    DKDefinitions(J).DKAddends = Selection.Value
    If DKDefinitions(J).DKAddends > 0 Then
        MComp = DKDefinitions(J).DKAddends
        I = 1
        AddendOffset = AddendNum + 1
        For C1 = AddendOffset To MComp + AddendNum
            Cells(R1, C1).Select
            DKDefinitions(J).DKComponents(I) = Selection.Value
            I = I + 1
        Next C1
    End If
```

```
'The 0th element of the DKDefinitions array will contain the
'total number of definitions!
```

```
DKDefinitions(0).DKNumber = J
J = J + 1
```

```
Next R1
```

```
End Sub
```

```
*****
'IndirectInit subroutine is called by the main PIGGYBACK routine to
'read the appropriate data from the inputs workbook and fill
'in public data arrays containing the components needing an indirect effect
*****
```

Sub IndirectInit()

```
Dim Result As Integer  
Dim LastRow As Long  
Dim IType As Integer
```

```
Result = 0  
'Activate the Inputs Workbook  
Call OpenWB(InputWorkbook, InputDirectory, Result)  
Workbooks(InputWorkbook).Activate  
Sheets(EffectsSheet).Activate
```

```
'Find the last row used  
Cells(65536, 2).Select  
Selection.End(xlUp).Select  
LastRow = Selection.Row
```

```
'zero out IndirectFactor array from last effect  
For R1 = 1 To MaxComps  
    IndirectFactor(R1) = 0  
Next
```

```
'Get the input data and write it to IndirectFactor and DistributionKey arrays
```

```
For R1 = 1 To LastRow  
    Compon = 0  
    IFact = 0  
    Cells(R1, ECompon).Select  
    If IsNumeric(Selection.Value) Then  
        Compon = Selection.Value  
        Cells(R1, ECompName).Select  
        If Selection.Value <> "NOT USED" Then  
            Cells(R1, EffectData(iEffect).InputCol).Select  
            If Selection.Value = "I" Then  
                IFact = 1  
            End If  
            If Selection.Value = "IM" Then  
                IFact = 2  
            End If  
        End If  
    End If  
    If Compon > 0 And IFact > 0 Then  
        IndirectFactor(Compon) = IFact  
        Cells(R1, DKNum).Select  
        If Selection.Value > 0 Then  
            DistributionKey(Compon) = Selection.Value  
        End If  
    End If  
End If  
Next R1  
  
End Sub
```

Sub MakeBookNames()

'This subroutine constructs all the appropriate workbook names based on Scenario, Year, and Effects

'ScenWB is the public variable name component for the scenario

'CurYear is the public variable containing the year

'EffectData(i).EffectLabel is the abbreviated effect label for each effect.

Dim iMBN As Integer

Dim iFPE As Integer

Dim PrevCode As String

'This sets the source for this year's Input Matrix to the OP Output Matrix

'May need to be changed to the MX Output Matrix

If iYear > 1 Then

 iMatrixBook = "FY" & Trim(strPrevYear) & EffectData(1).InputMatrix & "." & ScenWB & ".xls"

End If

InputWorkbook = InputTablesFile(iYear)

For iMBN = 1 To NumEffects

 EffectData(iMBN).WorkBookName = "FY" & Trim(strCurYear) & EffectData(iMBN).EffectLabel & "." & ScenWB & ".xls"

Next

End Sub

Sub InitializeWorkbookNames(ErrFlag)

'Populates an array for all possible years and all possible effects so that even previous
'year workbook names will be known

Dim iIWNYr, iIWNEf As Integer

On Error GoTo IWNErr

For iIWNYr = 1 To TotalYears

For iIWNEf = 1 To TotalEffects

AllWorkBookNames(iIWNYr, iIWNEf) = "FY" & Trim(YearLabel(iIWNYr)) &
EffectData(iIWNEf).EffectLabel & "." & ScenWB & ".xls"

Next

Next

IWNEResume:

On Error GoTo 0

Exit Sub

IWNErr:

ErrFlag = 1

Resume IWNEResume

End Sub

Sub GetInputMatrix()

'Copies the input matrix from the specified file.

'Copies from sheet IMatrixSheet in book IMatrixBook in directory IMatrixDir

'Copies to the current workbook to sheet 'InputMatrix'

'Variable definitions

'booValuesOpen True if we had to open the values workbook, False if it was already open

Dim booValuesOpen As Boolean

'Result return value from the OpenWB subroutine

Dim Result As Integer

'TargetBook name of the workbook copying to

'TargetSheet name of the sheet we were on in TargetBook when we arrived here

Dim TargetBook, TargetSheet As String

'PrevDir (optional) directory for the output matrix when starting from a year other than year 1

Dim PrevDir As String

'SourceSheet is the name of the worksheet from which data is taken. It is equal to

' IMatrixSheet, except that it is IMatrixStart for the first effect of the first year

Dim SourceSheet As String

TargetBook = ActiveWorkbook.Name

TargetSheet = ActiveSheet.Name

'Moved the following lines from MakeBookNames

**** TO CHECK FOR iYear > 1 and effect = 1, then

***** GO TO OutputMatrix as source

'So if start in middle of year IMatrixSheet won't be set to 'OutputMatrix'

If iYear > 1 And iEffect = 1 Then

 IMatrixSheet = "OutputMatrix"

End If

SourceSheet = IMatrixSheet

If iYear = 1 And iEffect = 1 Then

 SourceSheet = IMatrixStart

End If

PrevDir = IMatrixDir

If FirstYear And (iEffect = 1) Then

 If Len(Trim(PriorYearDir)) > 0 Then

 PrevDir = PriorYearDir

 End If

End If

'==== Part 1

'Test to see whether the specified workbook exists, has a worksheet with the right name, and
'appears to have data on that worksheet in the right format.

'1.1 Attempt to open the workbook

'If getting from MainBook, it must already be open

Result = 0

If IMatrixBook <> MainBook Then

 Call OpenWB(IMatrixBook, PrevDir, Result)

Else

```
Windows(MainBook).Activate
End If
booValuesOpen = False
If Result > 0 Then
    booValuesOpen = True
End If
If Result > 1 Then
    Msg = "The workbook you specified for the Input Matrix, " & IMatrixBook & " in " & IMatrixDir _
        & " did not already exist. It has now been created, but does not contain the input matrix." &
vbCrLf & vbCrLf _
        & "Copy the Input Matrix to this workbook, or change the name and/or location of the
workbook " _
        & "that contains the Input Matrix."
    MsgBox Msg, vbOKOnly + vbCritical, "Input File Error"
    Exit Sub
End If
'1.2 Does it have a worksheet with the right name?
'make sure it's not already open
Application.StatusBar = "Checking for Input Matrix worksheet"
Eflag = 0
'This will generate Error 9 (subscript out of range) if the specified worksheet does not exist
On Error GoTo LinkMatrixError
Eflag = 0
Sheets(SourceSheet).Activate
If Eflag > 0 Then
    Msg = "The specified worksheet name (" & SourceSheet & ") could not be found." & vbCrLf &
vbCrLf _
        & "Check the worksheet name, update the entry, and try again."
    MsgBox Msg, vbOKOnly + vbCritical, "Invalid worksheet name"
    GoTo EndProcess
End If
'===== End Part 1.

'==== Part 2
'Is there already a sheet named 'InputMatrix' in the target workbook?
'If so, delete it so it can be overwritten
Workbooks(TargetBook).Activate
Eflag = 0
On Error GoTo LinkMatrixError
Sheets(IMatrixSheet).Activate
On Error GoTo 0
'No error is returned if the sheet already exists. Delete it.
If Eflag = 0 Then
    Sheets(IMatrixSheet).Select
    Application.DisplayAlerts = False
    Sheets(IMatrixSheet).Delete
    Application.DisplayAlerts = True
End If
'Make sure there is not already a sheet with the same name as the (temporary) value of
'the Input Matrix about to be copied (probably OutputMatrix)
Eflag = 0
On Error GoTo LinkMatrixError
Sheets(FYIMatrix).Activate
On Error GoTo 0
If Eflag = 0 Then
    Sheets(FYIMatrix).Select
```

```
Application.DisplayAlerts = False  
Sheets(FYIMatrix).Delete  
Application.DisplayAlerts = True  
End If
```

```
'copy the old source to the target  
Windows(IMatrixBook).Activate  
Sheets(SourceSheet).Select  
Sheets(SourceSheet).Copy After:=Workbooks(TargetBook).Sheets(1)  
Workbooks(TargetBook).Activate  
Sheets(SourceSheet).Select  
Sheets(SourceSheet).Name = FYIMatrix
```

```
If IMatrixSheet = "OutputMatrix" Then  
    IMatrixSheet = "InputMatrix"  
End If
```

```
Exit Sub
```

```
EndProcess:  
'Exit here if there were errors  
'Allow the screen to be updated  
Application.ScreenUpdating = True  
'Turn control of the status bar back to Excel  
Application.StatusBar = False  
Eflag = 1  
On Error GoTo 0  
Exit Sub
```

```
'This error routine is invoked if there are errors at critical points during the  
'importing of the input matrix  
LinkMatrixError:  
Eflag = 1  
Resume Next
```

```
End Sub
```

```
'Initialize the CSName Array with the name for each cost segment.
```

Sub CSNameInit(ErrFlag)

```
Dim LastRow As Long
Dim CS As Long
Dim I As Long
Dim Name As String

On Error GoTo CSNInitErr

For I = 1 To MaxCS
    CSName(I) = ""
Next I

Sheets("CostSegments").Activate
'Find last cost segment on page
Cells(65536, 1).Select
Selection.End(xlUp).Select
LastRow = Selection.Row
For I = 2 To LastRow
    Cells(I, 1).Select
    CS = Selection.Value
    Cells(I, 2).Select
    Name = Selection.Value
    CSName(CS) = Name
Next I

CSNIEResume:
On Error GoTo 0
Exit Sub

CSNInitErr:
ErrFlag = 1
Resume CSNIEResume
End Sub
```

H. Utility Routines – Module1 and Module 2

Module1

' Various common routines

Sub FindText(Txt As String)

'Finds the text specified. Selected range must already be set

```
Selection.Find(What:=Txt, _  
    After:=ActiveCell, _  
    LookIn:=xlFormulas, _  
    LookAt:=xlPart, _  
    SearchOrder:=xlByRows, _  
    SearchDirection:=xlNext, _  
    MatchCase:=False, _  
    SearchFormat:=False).Select
```

End Sub

Sub Cleanup()

```
Call SaveCloseWorkbooks  
Windows(MainBook).Activate  
Application.StatusBar = "Saving " & ActiveWorkbook.Name  
Application.Calculation = xlAutomatic  
ActiveWorkbook.Save  
Application.StatusBar = False  
Application.ScreenUpdating = True  
End Sub
```

Sub SaveCloseWorkbooks()

'Saves and closes workbooks that the program had to open or create.

'Workbooks that were not opened by the program are not closed.

Dim iWB As Integer

If Not ynSaveClose Then

Exit Sub

End If

For iWB = 1 To SCCount

Application.StatusBar = "Checking " & Workbooks(SaveCloseNames(iWB)).Name

On Error GoTo CloseErr

Windows(SaveCloseNames(iWB)).Activate

On Error GoTo 0

If ActiveWorkbook.Saved Then

Application.StatusBar = "Closing " & Workbooks(SaveCloseNames(iWB)).Name

Else

Application.StatusBar = "Saving and closing " & Workbooks(SaveCloseNames(iWB)).Name

ActiveWorkbook.Save

End If

ActiveWorkbook.Close

NextBook:

On Error GoTo 0

Next

Exit Sub

CloseErr:

Resume NextBook

End Sub

Sub RefreshInputMatrix()

'Variable definitions

'R row pointer
'C column pointer
Dim R, C As Long

'NonNum non-numeric data counter
'ZCount zero data counter
'TotCount counter for total cells to copy
Dim NonNum, ZCount, TotCount As Long

'booValuesOpen True if we had to open the values workbook, False if it was already open
'booDataCheck True means perform data check of matrix. False means bypass check
Dim booValuesOpen, booDataCheck As Boolean

'ZRatio percent of cells with zero (or blank)
'ZLimit if ZRatio is less than ZLimit, user will have to confirm to continue
Dim ZRatio, ZLimit As Single

'Result return value from the OpenWB subroutine
Dim Result As Integer

ZLimit = 96
ThisBook = ActiveWorkbook.Name
ThisSheet = ActiveSheet.Name
'This prevents the screen from scrolling all over the place during processing
Application.ScreenUpdating = False

Msg = "This will copy the values in the (1600 by 200) matrix from sheet " & "InputMatrix" & " in "
_ & IMatrixDir & IMatrixBook _
_ & " to the 'InputMatrix' sheet in this workbook." & vbCrLf & vbCrLf & "Do you want to continue?"
If MsgBox(Msg, vbYesNo + vbQuestion + vbDefaultButton2, "Confirm action") = vbNo Then
 GoTo EndProcess
End If
Msg = ""

'==== Part 1

'Test to see whether the specified workbook exists, has a worksheet with the right name, and
'appears to have data on that worksheet in the right format.

'1.1 Attempt to open the workbook
Call OpenWB(IMatrixBook, IMatrixDir, Result)

booValuesOpen = False
If Result > 0 Then
 booValuesOpen = True

End If

If Result > 1 Then

 Msg = "The workbook you specified for the Input Matrix, " & IMatrixBook & " in " & IMatrixDir _
 _ & " did not already exist. It has now been created, but does not contain the input matrix." &

vbCrLf & vbCrLf _

 _ & "Copy the Input Matrix to this workbook, or change the name and/or location of the
workbook " _

 _ & "that contains the Input Matrix."

 MsgBox Msg, vbOKOnly + vbCritical, "Input File Error"

```
Exit Sub
End If
'1.2 Does it have a worksheet with the right name?
'make sure it's not already open
Application.StatusBar = "Checking for Input Matrix worksheet"
Eflag = 0
'This will generate Error 9 (subscript out of range) if the specified worksheet does not exist
On Error GoTo LinkMatrixError
Eflag = 0
Sheets(IMatrixSheet).Activate
If Eflag > 0 Then
    Msg = "The specified worksheet name (" & IMatrixSheet & ") could not be found." & vbCrLf &
    vbCrLf _
    & "Check the worksheet name, update the entry, and try again."
    MsgBox Msg, vbOKOnly + vbCritical, "Invalid worksheet name"
    GoTo EndProcess
End If

'===== End Part 1.

'copy the data
Range(Cells(1, 1), Cells((ROffSet + 1600), (COffset + 200))).Select
Selection.Copy
'Come back to this workbook and paste it
Workbooks(ThisBook).Activate
Worksheets(IMatrixStart).Activate
Cells(1, 1).Select
ActiveSheet.Paste
Application.CutCopyMode = False
'Put the cursor where we want it
Cells(1, 1).Select

'Go back to the worksheet with the button
Sheets(ThisSheet).Activate

EndProcess:
'Allow the screen to be updated
Application.ScreenUpdating = True
'Turn control of the status bar back to Excel
Application.StatusBar = False
On Error GoTo 0
MsgBox "Done refreshing Input Matrix"
Exit Sub

'This error routine is invoked if there are errors at critical points during the
'importing of the input matrix
LinkMatrixError:
Eflag = 1
Resume Next

End Sub
```

Public Sub ClassArrayInit(ErrFlag)

'Moved this code into its own subroutine so it is only done once,

Dim MCRCnt, MCCCnt As Long
Dim MCR1, MCC1 As Long
Dim I, J, R As Long

'Initializes the Mail CLasses array

'It is important that MaxClass has already been initialized before this Sub is executed
'because we check in here that no Mail Class has a value larger than MaxClass

'Go to the Master workbook and read in the Mail Class data
Workbooks(MainBook).Activate
Application.GoTo Reference:="MailClassRange"
'Identify the location and size of MailClassRange
MC = Names("MailClassRange").RefersToRange.Value
MCRcnt = UBound(MC, 1)
MCCCnt = UBound(MC, 2)
MCR1 = Selection.Row
MCC1 = Selection.Column

'This reads the entire range into the MailClass data structure

I = MCR1

Do Until I = (MCR1 + MCRcnt)

R = I - MCR1 + 1

For J = MCC1 To (MCC1 + MCCCnt - 1)

Cells(I, J).Select

Select Case J

Case 1

Classes(R).ClassName = Selection.Value

Case 2

If Not IsNumeric(Selection.Value) Then

ErrorCount = ErrorCount + 1

ReDim Preserve strErrorCheck(ErrorCount)

strErrorCheck(ErrorCount) = "Invalid Mail Class entry on line " & R

Elseif Selection.Value > MaxClass Then

ErrorCount = ErrorCount + 1

ReDim Preserve strErrorCheck(ErrorCount)

strErrorCheck(ErrorCount) = "Mail Class (" & Selection.Value & ") on line " & I & " is larger
than maximum (" & MaxClass & ")"

Else

Classes(R).Component = Selection.Value

End If

Case 3

Classes(R).Level = Selection.Value

Case 4

Classes(R).Parent = Selection.Value

Case 5

Classes(R).SubTot = Selection.Value

Case 6

Classes(R).TotOnly = Selection.Value

Case 7

Classes(R).Header = Selection.Value

Case 8

Classes(R).Sort = Selection.Value

```
Case 9
  Classes(R).Display = Selection.Value
End Select
Next
I = I + 1
Loop
ClassMax = R
Exit Sub

CAInitErr:
ErrFlag = 1
Resume Next

End Sub
```

Public Sub GetClasses()

'Inserts the Mail Classes onto the current sheet in the location specified
'by public variables Row1, Col1

'Dim CSSheet, CSBook As String
Dim I As Long
Dim MC As Variant
Dim Msg, Prefix As String

For I = 1 To ClassMax
R = Row1 + I
Cells(R, Col1).Select
Selection.Value = I
Cells(R, Col1 + 1).Select
Prefix = ""
If Classes(I).Display = 2 Or Classes(I).Display = 4 Then
Prefix = " "
Elseif Classes(I).Display = 3 Then
Prefix = " "
End If
Selection.Value = Prefix & Classes(I).ClassName
If Classes(I).Display = 1 Or Classes(I).Display = 4 Then
Selection.Font.Bold = True
End If

If Classes(I).Component > 0 Then
Cells(R, Col1 + 2).Select
Selection.Value = Classes(I).Component
If CSSheetMode = "VolAdj" Then
For J = 1 To MaxClass
If VolAdjClass(J) = Classes(I).Component Then
Cells(R, Col1 + 3).Select
Selection.Value = VolAdjFactor(J)
End If
Next
End If
End If
Next

If CSSheetMode = "VolAdj" Then
Cells(Row1 - 4, Col1 + 3).Select
Selection.Value = "Mail Volume Factor"
Selection.WrapText = True
Columns(5).Select
Selection.ColumnWidth = 10
End If

'8-3-04 NK. Reduce all Row1 - values by one to eliminate unused row
'and add row at top of classes with label
'8/04/05 - put the rows back!

Cells(Row1 - 3, Col1 + 1).Select
Selection.Value = "Component"
If CSSheetMode = "VolAdj" And Left(CSSheet, 2) = "CS" Then

```
Cells(Row1 - 2, Col1 + 1).Select
Selection.Value = "MV Type"
Elseif CSSheetMode = "ME" And Left(CSSheet, 2) = "CS" Then
Cells(Row1 - 2, Col1 + 1).Select
Selection.Value = "Factor"
Cells(Row1 - 1, Col1 + 1).Select
Selection.Value = "Classes"
Elseif CSSheet = "Ratios" Then
Cells(Row1 - 2, Col1 + 1).Select
Selection.Value = "Usage"
End If
Cells(Row1, Col1 + 1).Select
Selection.Value = "Mail Class, Subclass, or Special Service"

Columns("A:C").Select
Columns("A:C").EntireColumn.AutoFit

Cells(1, 1).Select

End Sub
```

Sub MakeOutputMatrix()

Dim R1, R2, C1, C2 As Long

Dim LastRow As Long

Dim CS As Byte

Dim Comp, Class As Integer

Dim LastCol As Integer

Dim OutMatrixSheetName, CSSheet As String

Dim LineFormula As String

Dim Mtx(200, 1600) As Double

Dim ColOffset As Integer

'Assumes we are already in the right workbook.

'Test to see if we already have an "OutputMatrix" sheet

'This will generate Error 9 (subscript out of range) if the specified worksheet does not exist

OutMatrixSheetName = "OutputMatrix"

Eflag = 0

On Error GoTo OutputMatrixError

Sheets(OutMatrixSheetName).Activate

On Error GoTo 0

'If it is already there, delete it

If Eflag = 0 Then

 Application.DisplayAlerts = False

 Sheets(OutMatrixSheetName).Delete

 Application.DisplayAlerts = True

End If

'Now create the sheet and move it to the end

Sheets.Add

ActiveSheet.Name = OutMatrixSheetName

J = Sheets.Count

Sheets(OutMatrixSheetName).Move After:=Sheets(J)

'Read the input matrix into memory array InMtx

'If reading from column less than InputClassStart, just set to zero

'InputClassStart is a constant, declared in ModPublicVars

Sheets(IMatrixSheet).Activate

For C1 = 1 To 200

 Application.StatusBar = StatusLabel & ": Reading Input Matrix " & C1 & " of 200"

 For R1 = 1 To 1600

 Mtx(C1, R1) = 0

 If C1 >= InputClassStart Then

 Cells(R1 + ROffset, C1 + COffset).Select

 Mtx(C1, R1) = Selection.Value

 End If

 Next

Next

'Cycle through the CS sheets, adding the values to the array

ColOffset = 3

If CSSheetMode = "VolAdj" Then

 ColOffset = 4

End If

For CS = 1 To 99

 Eflag = 0

 CSSheet = "CS" & Right("00" & CStr(CS), 2)

 Application.StatusBar = StatusLabel & ": Adding " & CSSheet & " to input values."

 On Error GoTo OutputMatrixError

```
'Generates an error if the sheet does not exist
  Sheets(CSSheet).Activate
  If Eflag = 0 Then
'Find the last row of data
  Cells(65536, Col1).Select
  Selection.End(xlUp).Select
  LastRow = Selection.Row
'Find the last column of data
  Cells(Row1 - 3, 255).Select
  Selection.End(xlToLeft).Select
  LastCol = Selection.Column
  For C1 = Col1 + ColOffset To LastCol
    Cells(Row1 - 3, C1).Select
    Comp = 0
    If IsNumeric(Selection.Value) Then
      Comp = Selection.Value
    End If
    If Comp > 0 Then
      For R2 = Row1 + 1 To LastRow
        Class = 0
        Cells(R2, Col1 + 2).Select
        If IsNumeric(Selection.Value) Then
          Class = Selection.Value
        End If
        Cells(R2, C1).Select
        If IsNumeric(Selection.Value) Then
          Mtx(Class, Comp) = Mtx(Class, Comp) + Selection.Value
        End If
      Next
    End If
  Next
End If
Next
'Make the first row tall enough to display the label in cell A1
  Rows(1).Select
  If Selection.RowHeight < 47 Then
    Selection.RowHeight = 47
  End If
  Cells(2, 1).Select
End If
Next

'Write the results to the output matrix
Sheets(OutMatrixSheetName).Activate
For C1 = 1 To 200
  Application.StatusBar = StatusLabel & ": Writing Output Matrix " & C1 & " of 200"
  For R1 = 1 To 1600
    If Mtx(C1, R1) <> 0 Then
      Cells(R1 + ROffset, C1 + COffset).Select
      Selection.Value = Mtx(C1, R1)
      Selection.NumberFormat = "#,##0"
    End If
  Next
Next
Next

Exit Sub

OutputMatrixError:
```

Eflag = 1
Resume Next

End Sub

Sub Auto_Open()

'This subroutine runs automatically when the workbook is opened

```
'Select the page with the buttons  
Worksheets("RFMenuPage").Activate  
'Turn off the Row/Column headings  
ActiveWindow.DisplayHeadings = False  
'Turn off the formula bar  
Application.DisplayFormulaBar = False  
ActiveWindow.WindowState = xlMaximized  
'Zoom to make the buttons and input area take up the entire screen  
Range("A1:J26").Select  
ActiveWindow.Zoom = True  
Application.GoTo Reference:="Version"  
Selection.Value = "Release 2a (20050117.1132)"  
MsgDisplay = False  
Call UpdateScenarioList  
MsgDisplay = False  
Call UpdateEffectsLists  
MsgDisplay = False  
Call UpdateYearsLists  
End Sub
```

'AddComps puts component names and numbers onto the already open template workbook
'in the order they appear on the Master Component list.

Sub AddComps()

'Index into Components array
Dim I As Long

'Component number for current element of Components array.
Dim Comp As Long

'Cost segment for the component for current element of Components array.
Dim CS As Long

'Component name for the component for current element of Components array.
Dim strCompName As String

'A string to represent the cost segment value.
Dim strCS As String

'The worksheet name for this cost segment.
Dim SheetName As String

'Contains last column number on worksheet.
Dim LastCol As Long

'Read through components array and find all non-subtotal components and their
'corresponding cost segment. Add to the next empty row in the cost segment workbook.

'Edited 3/12/04 to add subtotal components as well

On Error GoTo 0

For I = 1 To ComponentCount

 'Use non-subtotal components only
 'or subtotal components for cost-containing cost segments
 'i.e., don't add subtotals on distribution key cost segments
 If Components(I).CompSubTot = False Or _
 (Componentes(I).CompCS <= LastCostCS And Components(I).CompSubTot = True) Then
 Comp = Components(I).CompNumber
 CS = Components(I).CompCS
 strCompName = Components(I).CompName

 'Open the Cost Segment worksheet for this component

 If CS <= LastCostCS Then
 strCS = Right("00" & Trim(Str(CS)), 2)
 SheetName = "CS" & strCS
 Sheets(SheetName).Activate
 'Find last used column
 'Changed 8/4/04 to use row with component names, instead of hard-code row 1
 Cells(ComponentRow - 1, 255).Select
 Selection.End(xlToLeft).Select
 LastCol = Selection.Column
 If LastCol < ComponentOffset - 1 Then
 LastCol = ComponentOffset - 1
 End If

```
Columns>LastCol + 1).Select
Selection.ColumnWidth = 14
Cells(ComponentRow - 1, LastCol + 1).Select
Selection.WrapText = True
Selection.Value = strCompName
Selection.HorizontalAlignment = xlCenter
If Components(l).CompSubTot = True Then
    Selection.Font.Bold = True
End If
Cells(ComponentRow, LastCol + 1).Select
Selection.Value = Comp
Selection.HorizontalAlignment = xlCenter
If Components(l).CompSubTot = True Then
    Selection.Font.Bold = True
End If
Rows(ComponentRow - 1).RowHeight = 80
'End If
End If
Next l

End Sub
```

Module2

'This module contains common subroutines and functions that are called by other subroutines and functions

Sub OpenWB(OpenBook, OpenPath, Result As Integer)

'New argument, IMode, to be added, but not functioning yet
'Sub OpenWB(OpenBook, OpenPath, Result As Integer, IMode As Integer)

'Input argument OpenBook is the name of the workbook to be opened.

'Input argument OpenPath is the path where the workbook to be opened is located.

'Output argument Result is set as follows:

- ' 0 = workbook was already open before calling
- ' 1 = workbook already existed, and is now open
- ' 2 = workbook had to be created

'Input argument IMode has the following meanings:

- ' 0 = create the workbook if it does not exist
- ' 1 = open it if it's not open, but do not create it if it does not exist
- ' 2 = activate it if it is already open, but do not open it

'In all cases, the workbook will be open and will be the active workbook upon exiting this sub

'Fill holds the next available file number

'EFlag is an error flag

Dim Fill, Eflag As Integer

'FullBookName is the concatenated path and workbook name

Dim FullBookName As String

'OldStatusText is the text that was displayed in the statusbar before coming here

Dim OldStatusText As String

Eflag = 0

Result = 0

If Len(OpenPath) > 0 Then

 FullBookName = OpenPath & OpenBook

Else

 FullBookName = DefaultDirectory & OpenBook

End If

'Check to see whether the file needs to be created

'Try to open the workbook. This generates an error (which is trapped) if it doesn't exist

'This uses a DOS-style Open statement rather than an Excel (Workbooks.Open) to test whether the file exists. FreeFile is a function that returns the next available file number.

OldStatusText = Application.StatusBar

Application.StatusBar = strCurYear & ": Checking for file " & FullBookName

Fill = FreeFile

On Error GoTo OpenWBErr

Open FullBookName For Input As Fill

Close Fill

On Error GoTo 0

If Eflag > 0 Then

 Workbooks.Add

 ActiveWorkbook.SaveAs Filename:=FullBookName, _
 FileFormat:=xlNormal, _

```
Password:="", _  
WriteResPassword:="", _  
ReadOnlyRecommended:=False, _  
CreateBackup:=False
```

```
Result = 2  
Else
```

```
'If we get here, the file exists. Check to see if it is already open  
'This will generate Error 9 (subscript out of range) if the workbook is not open  
'Note that this routine will not distinguish between the desired workbook and  
'one with the same name from a different folder that is already open.  
'If there is another workbook with the same name already open, this routine will  
'think that the desired workbook is open already.  
On Error GoTo OpenWBErr  
Windows(OpenBook).Activate  
On Error GoTo 0  
If Eflag > 0 Then  
Application.StatusBar = strCurYear & ": Opening " & FullBookName  
'The second paramter of the Open method (False) tells Excel NOT TO UPDATE LINKS  
'Of course, there shouldn't be any links in these workbooks.  
Workbooks.Open FullBookName, False  
Result = 1  
End If  
End If
```

```
OpenSave:
```

```
'All exits from the Sub go through here to track whether to add workbook to  
'list of workbooks that need to be saved and closed at the end  
If Result > 0 Then  
SCCCount = SCCCount + 1  
SaveCloseNames(SCCCount) = OpenBook  
End If
```

```
Application.StatusBar = OldStatusText  
Exit Sub
```

```
OpenWBErr:  
Eflag = 1  
Resume Next
```

```
End Sub
```

Sub AddCSSheet()

'Adds Cost Segment worksheets and places Mail Class labels on each sheet
'Make sheet name for each Cost Segment
'Also makes one named "Ratios"

Dim LastRow As Long
Dim SheetName As String
Dim Eflag As Integer
Dim WorkingBook As String

WorkingBook = ActiveWorkbook.Name
Workbooks(MainBook).Activate
Worksheets(ComponentSheet).Activate
Cells(65535, 5).Select
Selection.End(xlUp).Select
LastRow = Selection.Row

*****NOTE ADDED 8/5/04 NEED TO CHANGE HARD-CODED COLUMN
*****REFERENCES TO USER DEFINED

For CS = 1 To MaxCS
Workbooks(MainBook).Activate
Worksheets(ComponentSheet).Activate
R = 2
booLoop = False
Do Until R > LastRow
Cells(R, 3).Select
If Selection.Value = 1 Then
Cells(R, 4).Select
If Val(Selection.Value) = CS Then
booLoop = True
Exit Do
End If
End If
R = R + 1
Loop

'This is to skip Cost Segments that do not exist
If booLoop Then

strCS = Right("00" & Trim(Str(CS)), 2)
SheetName = "CS" & strCS
Workbooks(WorkingBook).Activate
'This will generate Error 9 (subscript out of range) if the specified worksheet does not exist
On Error GoTo NoSheet
Eflag = 0
Sheets(SheetName).Activate
On Error GoTo 0
If Eflag > 0 Then
Sheets.Add
ActiveSheet.Name = SheetName
J = Sheets.Count
Sheets(SheetName).Move After:=Sheets(J)
Else

```
'This makes sure there's nothing on the page before we start  
Cells.Select  
Selection.ClearContents  
End If
```

```
'Put a label in the top left corner (Cell A1)
```

```
*****  
**** SECTION MODIFIED 3/16/04 BY N KAY TO ADD COST SEGMENT ***  
**** NAME TO SHEET LABEL AND TO FORMAT FOR PRINTING ***  
**** DON'T DO FORMATTING FOR PRINTING NOW BECAUSE IT WHITES ***  
**** OUT THE SCREEN. SAVE IT FOR PRINTING REPORTS ***  
*****
```

```
CostSegName = CSName(CS)  
Range("A1:C1").Select  
Selection.MergeCells = True  
Selection.WrapText = True  
'New header added 8/4/04 to conform to report standards  
If NumEffects > 1 And EffectData(iEffect).EffectLabel = "MX" Then  
Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & _  
"Change Report" & Chr(10) & "Table " & NumEffects + 2 & ". " & EffectLabel & Chr(10) &  
"C/S " & CS & " " & CostSegName  
Else  
Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & _  
"Change Report" & Chr(10) & "Table " & iEffect + 1 & ". " & EffectLabel & Chr(10) & _  
"C/S " & CS & " " & CostSegName  
End If  
'Selection.Value = EffectLabel & Chr(10) & "CS " & CS & " " & CostSegName & Chr(10) & "FY  
& strCurYear  
Selection.Font.Color = vbBlue  
Selection.Font.Bold = True  
Selection.Font.Size = 12  
Rows("1:1").RowHeight = 70
```

```
'Put the mail classes onto this sheet  
'On Error GoTo 0  
CSSheet = SheetName  
Call GetClasses  
Columns("A:C").Select  
Columns("A:C").EntireColumn.AutoFit
```

```
End If  
Next
```

```
'Add the Ratios sheet  
SheetName = "Ratios"  
Workbooks(WorkingBook).Activate  
On Error GoTo NoSheet  
Eflag = 0  
Sheets(SheetName).Activate  
On Error GoTo 0  
If Eflag > 0 Then  
Sheets.Add  
ActiveSheet.Name = SheetName
```

```
J = Sheets.Count
Sheets(SheetName).Move After:=Sheets(J)
Else
'This makes sure there's nothing on the page before we start
  Cells.Select
  Selection.ClearContents
End If
Range("A1.C1").Select
Selection.MergeCells = True
Selection.Value = ScenDescription & " - FY" & strCurYear & Chr(10) & _
  "Component Change Report" & Chr(10) & EffectLabel & Chr(10) & _
  "Ratios"
'Selection.Value = EffectLabel & Chr(10) & " Ratios" & Chr(10) & "FY " & strCurYear
Selection.Font.Color = vbBlue
Selection.Font.Bold = True
Selection.Font.Size = 12
Rows("1:1").RowHeight = 70

CSSheet = SheetName
Call GetClasses

Columns("A:C").Select
Columns("A:C").EntireColumn.AutoFit

Exit Sub

NoSheet:
Eflag = 1
Resume Next

End Sub
```

Sub FindChildren(Cmpt, FCMode As Integer, OutArray)

'This Sub finds all the children of component Cmpt
'Input argument Cmpt specifies which component to find the children of
'Input argument FCMode indicates:
' 1 = Find the immediate children (next level only)
' 2 = Find all descendants
'Argument OutArray must be declared in the calling routine and will have the following info when it leaves here:
' OutArray(0) holds the number of elements
' OutArray(i) holds the ith component number that is a child of Cmpt, where i <= OutArray(0)

```
Dim FCArray(1000), XArray(1000) As Long
Dim I, J, X, Y, Z As Integer
Dim ChildFlag, CurComp As Boolean
Dim Pass As Integer
```

```
For I = 1 To 1000
    FCArray(I) = 0
    XArray(I) = 0
Next
OutArray(0) = 0
X = 0
```

```
'Start by finding all the immediate children of Cmpt
For I = 1 To ComponentCount
    If Components(I).CompParent = Cmpt Then
'X contains the current count of the number of child components
        X = X + 1
        FCArray(X) = Components(I).CompNumber
    End If
Next
```

```
'For Mode 1 simply load these values into OutArray and return
If FCMode = 1 Then
    OutArray(0) = X
    For I = 1 To X
        OutArray(I) = FCArray(I)
    Next
    Exit Sub
End If
```

```
'For Mode 2, look for additional levels of children
'If any are found, their parent is removed from the array, and children added to the end
'Keep repeating until we can go through the loop without finding any more children
If FCMode = 2 Then
    Pass = 0
    Do
        Z = X
        Pass = Pass + 1
        ChildFlag = False
        For I = 1 To Z
            CurComp = False
            For J = 1 To ComponentCount
                If Components(J).CompParent = FCArray(I) Then
                    XArray(I) = FCArray(I)
                    ChildFlag = True
                End If
            Next J
        Next I
    Loop While ChildFlag
```

```
    CurComp = True
    X = X + 1
    FCArray(X) = Components(J).CompNumber
End If
Next
If CurComp Then
    FCArray(I) = -1
End If
Next
Loop While ChildFlag = True
'Copy the array to OutArray, deleting the parents whose children are on the list
J = 0
For I = 1 To X
    If FCArray(I) <> XArray(I) And FCArray(I) > 0 Then
        J = J + 1
        OutArray(J) = FCArray(I)
    End If
Next
OutArray(0) = J
Exit Sub
End If

'+++++
' This should probably be deleted during final cleanup
'+++++
MsgBox "This is unexpected!" & vbCrLf & vbCrLf & "Sub FindChildren was called with FCMode =
" & FCMode _
    & vbCrLf & vbCrLf & "Only modes 1 and 2 have meaning.", vbOKOnly + vbExclamation,
"OOPS!"

End Sub
```

Sub FindPreviousEffect()

```
'Looks at the value of the element 'InputMatrix' and 'iEffect'  
'to determine the value of InputEffect  
'(the effect whose output matrix is the current effect's input matrix)  
Dim PrevCode As String  
Dim iFPE As Integer  
InputEffect = 0  
PrevCode = EffectData(iEffect).InputMatrix  
For iFPE = 1 To NumEffects  
    If EffectData(iFPE).EffectLabel = PrevCode Then  
        InputEffect = iFPE  
        Exit For  
    End If  
Next  
End Sub
```

Sub ScenarioChange()

'Updates the Scenario description when the user changes the scenario selection

Dim Scn As Long

Dim Scenar As String

Dim S1 As String

S1 = ActiveSheet.Name

Application.GoTo Reference:="ScenarioIndex"

Scn = Selection.Value

Application.GoTo Reference:="ScenariosSheet"

ScenarioRow = Scn + 1

Cells(ScenarioRow, 3).Select

Scenar = Selection.Value

Application.GoTo Reference:="ScenarioLabel"

Selection.Value = Scenar

Application.GoTo Reference:="ScenarioIndex"

End Sub

Sub MultiYearOptionClick()

'Runs when the user clicks the multi-year option (radio) button

ynMultiYear = True

Application.GoTo Reference:="SingleYearData"

Selection.Font.Bold = False

Selection.Font.ColorIndex = 48

Application.GoTo Reference:="SingleYearLabel"

Selection.Font.Bold = False

Selection.Font.ColorIndex = 48

Application.GoTo Reference:="MultiYearData"

Selection.Font.Bold = True

Selection.Font.ColorIndex = 1

Application.GoTo Reference:="MultiYearLabel"

Selection.Font.Bold = True

Selection.Font.ColorIndex = 1

Application.GoTo Reference:="ScenarioIndex"

End Sub

Sub SingleYearOptionClick()

'Runs when the user clicks the single-year option (radio) button

ynMultiYear = False

Application.GoTo Reference:="SingleYearData"

Selection.Font.Bold = True

Selection.Font.ColorIndex = 1

Application.GoTo Reference:="SingleYearLabel"

Selection.Font.Bold = True

Selection.Font.ColorIndex = 1

Application.GoTo Reference:="MultiYearData"

Selection.Font.Bold = False

Selection.Font.ColorIndex = 48

Application.GoTo Reference:="MultiYearLabel"

Selection.Font.Bold = False

Selection.Font.ColorIndex = 48

Application.GoTo Reference:="ScenariolIndex"

End Sub

Sub CheckEffectOrder()

'Makes sure the Start Effect is not after the End Effect

Application.GoTo Reference:="StartEffect"

iStartEffect = Selection.Value

Application.GoTo Reference:="EndEffect"

iEndEffect = Selection.Value

If iStartEffect > iEndEffect Then

 MsgBox "Start Effect and End Effect are in the wrong order.", vbOKOnly + vbExclamation,
 "Invalid Input"

End If

End Sub

Sub CheckYearOrder()

'Makes sure the Start Year is not after the End Year

Application.GoTo Reference:="AnalysisStartYear"

iStartYear = Selection.Value

Application.GoTo Reference:="AnalysisEndYear"

iEndYear = Selection.Value

If iStartYear > iEndYear Then

 MsgBox "Start Year and End Year are in the wrong order.", vbOKOnly + vbExclamation, "Invalid
Input"

End If

End Sub

Sub UpdateLists()

'Updates the scenario list and the Effects Lists (From and To) on the main worksheet

MsgDisplay = False

Call UpdateScenarioList

MsgDisplay = False

Call UpdateEffectsLists

MsgDisplay = False

Call UpdateYearsLists

MsgBox "Scenario, Effects, and Years lists have been refreshed.", vbOKOnly + vbInformation,

"Action complete"

End Sub

Sub UpdateScenarioList()

Dim R As Long

Dim PrevSht As String

PrevSht = ActiveSheet.Name

'Prevent the screen from scrolling all over the place

Application.ScreenUpdating = False

'Find the last scenario and populate the drop-down list

Worksheets("Scenarios").Activate

Range("B1").Select

Selection.End(xlDown).Select

R = Selection.Row

Worksheets(PrevSht).Activate

ActiveSheet.Shapes("Drop Down 19").Select

Selection.ListFillRange = "Scenarios!\$B\$2:\$B\$" & Trim(Str(R))

Application.GoTo Reference:="ScenariIndex"

If MsgDisplay Then

 MsgBox "Scenario list updated.", vbOKOnly + vbInformation, "Action completed"

End If

MsgDisplay = True

End Sub

Sub UpdateEffectsLists()

Dim R As Long

Dim PrevSht As String

PrevSht = ActiveSheet.Name

'Prevent the screen from scrolling all over the place

Application.ScreenUpdating = False

'Find the last Effect and populate the drop-down lists

Worksheets("EffectsParameters").Activate

Range("A1").Select

Selection.End(xlDown).Select

R = Selection.Row

TotalEffects = R - 1

Worksheets(PrevSht).Activate

'Drop Down 30 is the "From" effect

'ActiveSheet.Shapes("Drop Down 30").Select

'Selection.ListFillRange = "EffectsParameters!\$A\$2:\$A\$" & Trim(Str(R))

'Drop Down 31 is the "To" effect

'ActiveSheet.Shapes("Drop Down 31").Select

'Selection.ListFillRange = "EffectsParameters!\$A\$2:\$A\$" & Trim(Str(R))

Application.GoTo Reference:="ScenarioIndex"

If MsgBox Then

 MsgBox "Effects lists updated.", vbOKOnly + vbInformation, "Action completed"

End If

MsgDisplay = True

End Sub

Sub UpdateYearsLists()

Dim R As Long

Dim PrevSht, YSht As String

PrevSht = ActiveSheet.Name

'Prevent the screen from scrolling all over the place

Application.ScreenUpdating = False

'Find the last Effect and populate the drop-down lists

Application.GoTo Reference:="InputTablesSheet"

YSht = ActiveSheet.Name

Selection.End(xlDown).Select

R = Selection.Row

TotalYears = R - 1

Worksheets(PrevSht).Activate

'Drop Down 32 is the "From" year

ActiveSheet.Shapes("Drop Down 32").Select

Selection.ListFillRange = YSht & "!\$A\$2:\$A\$" & Trim(Str(R))

'Drop Down 33 is the "To" year

ActiveSheet.Shapes("Drop Down 33").Select

Selection.ListFillRange = YSht & "!\$A\$2:\$A\$" & Trim(Str(R))

'Drop Down 34 is the single year

ActiveSheet.Shapes("Drop Down 34").Select

Selection.ListFillRange = YSht & "!\$A\$2:\$A\$" & Trim(Str(R))

Application.GoTo Reference:="ScenarioIndex"

If MsgBox "Years lists updated.", vbOKOnly + vbInformation, "Action completed"

End If

MsgDisplay = True

End Sub

Public Function IsClass(C) As Boolean

'The argument, C, is a variant.

'If C is numeric and is the value of a valid mail class, the function returns True.

'Otherwise it returns False

Dim I As Integer

IsClass = False

If Not IsNumeric(C) Or IsEmpty(C) Then 'Nancy added test for empty cell 10/5/04

Exit Function

End If

For I = 1 To MaxClass

If C = Classes(I).Component Then

IsClass = True

Exit For

End If

Next

End Function

Public Function IsComponent(C) As Boolean

'The argument, C, is a variant.

'If C is numeric and is the value of a valid component, the function returns True.

'Otherwise it returns False

Dim I As Integer

IsComponent = False

If Not IsNumeric(C) Then

Exit Function

End If

For I = 1 To ComponentCount

If C = Components(I).CompNumber Then

IsComponent = True

Exit For

End If

Next

End Function

I. Roll Forward Distribution Key Loader Module

The roll forward distribution keys are entered into the Base Year A cost matrix workbook using a separate Excel/VBA module called RFLoader.xls. The VBA program for this module is contained in this section.

'GLOBAL VARIABLES AND CONSTANTS USED IN ROLL FORWARD LOADER MODULE

'Offsets for input matrix

Public Const ROffset As Integer = 3

Public Const COffset As Integer = 3

'These constants are the columns for the data on the EffectsList worksheet

Public Const EComp As Integer = 2

Public Const ECompName As Integer = 3

Public Const EMV As Integer = 5

'These constants are the columns for the data on the Components worksheet

Public Const CComp As Integer = 1

Public Const CCompName As Integer = 2

Public Const CLevel As Integer = 3

Public Const CCS As Integer = 4

Public Const CParent As Integer = 5

Public Const CSubtotal As Integer = 6

Public Const CTotOnly As Integer = 7

'These constants are the locations for the data on the RFKeys worksheet

Public Const RFTitle As Integer = 1

Public Const RFNewComp As Integer = 2

Public Const RFOldComp As Integer = 3

Public Const RFMailVol As Integer = 5

Public Const RFFirstClass As Integer = 10

Public Const RFFirstCol As Integer = 5

Public Const RFClassCol As Integer = 3

'First and last class numbers holding cost data

Public Const StartClass As Integer = 101

Public Const LastClass As Integer = 200

'Maximum number of roll forward keys that can be updated

Public Const MaxRFKeys As Integer = 75

'Lowest possible component number for roll forward distribution keys

Public Const LowestRFKey As Long = 1400

'Public declarations for variables to hold A Matrix file and directory,

'Effects worksheet file and directory

'Components worksheet file and directory

'These variables will be loaded in the Init subroutine

Public AFile As String

Public ADir As String

Public AOut As String

Public EFile As String

Public EDir As String

Public EOut As String

Public CFile As String
Public CDir As String
Public COut As String
Public USPS As Boolean

'Holds main workbook and worksheet name
Public ThisBook As String
Public ThisSheet As String

'Name of various worksheets
Public EffectsSheet As String
Public USPSSheet As String
Public PRCSheet As String
Public IMSheet As String
Public MenuPage As String
Public RFKeySheet As String

'Response to various calls
Public Response As Integer

'These variables are used to get the column and row in the input matrix
'from the input component and class number.
Public Comp As Long
Public Class As Long
Public IMRow As Long
Public IMCol As Long

'Variables to hold the component numbers and names of the new keys
Public RFNewKeys(MaxRFKeys) As Long
Public RFTitles(MaxRFKeys) As String
Public RFMV(MaxRFKeys) As Boolean

'Function to get input matrix address, can easily be substituted for a new
'one when flexible addressing is implemented.

'9/23/04 - Offset row and column to match RF Model.

Public Function IMAAddress(Comp, Class, IMRow, IMCol)

IMRow = Comp + ROffSet
IMCol = Class + COffSet

End Function

Main subroutine that runs when the "GO" button is pressed on the Menu worksheet

Sub Menu()

Dim LoadFlag, WriteAFlag, EffectsFlag, CompPRCFlag, CompUSPSFlag As Boolean

```
MenuPage = "LoaderMenu"  
RFKeySheet = "RFKeys"
```

```
Application.GoTo Reference:="LoadA"  
LoadFlag = Selection.Value  
Application.GoTo Reference:="WriteA"  
WriteAFlag = Selection.Value  
Application.GoTo Reference:="UpdateE"  
EffectsFlag = Selection.Value  
Application.GoTo Reference:="UpdatePRC"  
CompPRCFlag = Selection.Value  
Application.GoTo Reference:="UpdateUSPS"  
CompUSPSFlag = Selection.Value
```

```
Call Init(Response)  
'Load A Matrix
```

```
Application.ScreenUpdating = False
```

```
If LoadFlag = True Then LoadA  
'Write Updated A Matrix to File  
If WriteAFlag = True Then WriteAll  
'Update Effects sheet in input workbook  
If EffectsFlag = True Then Update_Effects  
'Update USPS and/or PRC components list  
If CompPRCFlag = True Then  
    USPS = False  
    Update_Components (USPS)  
End If  
If CompUSPSFlag = True Then  
    USPS = True  
    Update_Components (USPS)  
End If  
If CompPRCFlag = True Or CompUSPSFlag = True Then  
    WriteComponents  
End If  
Application.ScreenUpdating = True  
End Sub
```

'This subroutine loads the A Matrix file into the roll forward
'distribution key update workbook so that it can be updated with new keys.

Sub LoadA()

Dim Eflag As Integer
Dim Result As Integer

Application.StatusBar = "Roll Forward Distribution Key Update: Loading A Matrix into Workbook"
ThisBook = ActiveWorkbook.Name
ThisSheet = ActiveSheet.Name

Call OpenWB(AFile, ADir, Result)
Eflag = 0
On Error GoTo LinkTemplateError

'Copy the data from the A Matrix file. Assume there is only one worksheet
'in this workbook!

If Eflag = 0 Then

Cells.Select
'Range("A1:GR1601").Select
'Range("GR1601").Activate
Selection.Copy
End If

Workbooks(ThisBook).Activate
Sheets(IMSHEET).Select
Range("A1").Select
ActiveSheet.Paste
Application.DisplayAlerts = False
Windows(AFile).Close SaveChanges:=False
Application.DisplayAlerts = True
Sheets(MenuPage).Activate
Application.StatusBar = "Roll Forward Distribution Key Update: Waiting for Next Command"

Exit Sub

LinkTemplateError:
Eflag = 1
Resume Next

End Sub

'This subroutine controls writing the new roll forward distribution keys to the
'A matrix

Sub WriteAll()

Application.StatusBar = "Roll Forward Distribution Key Update: Writing Updated A Matrix to File"

If Response > 0 Then

 Sheets(MenuPage).Activate

 'Cells(1, 1).Select

 Exit Sub

End If

CheckA

If Response > 0 Then

 Sheets(MenuPage).Activate

 'Cells(1, 1).Select

 Exit Sub

End If

WriteA

Application.StatusBar = "Roll Forward Distribution Key Update: Waiting for Next Command"

End Sub

'Reads in the names and numbers for all of the new keys from the 'RFKeys' worksheet
'Also reads in all of the input files and directories

Sub Init(Response)

```
Dim ThisComp As Long  
Dim ThisName As String  
Dim ThisMV As String  
Dim I As Integer
```

```
'Read in the A matrix file name and directory  
Worksheets("Locations").Activate  
Application.GoTo Reference:="AFileName"  
AFile = Selection.Value  
Application.GoTo Reference:="ADirectory"  
ADir = Selection.Value  
Application.GoTo Reference:="AOutput"  
AOut = Selection.Value  
Application.GoTo Reference:="ISheet"  
IMSheet = Selection.Value
```

```
'Put a backslash on the directory if the user left it off  
If Right(ADir, 1) <> "\" Then  
    ADir = ADir & "\"  
End If
```

```
'Read in the Effects workbook file and directory  
Application.GoTo Reference:="EFileName"  
EFile = Selection.Value  
Application.GoTo Reference:="EDirectory"  
EDir = Selection.Value  
Application.GoTo Reference:="ESheet"  
EffectsSheet = Selection.Value  
If EDir = "" Or EDir = " " Then  
    EDir = ADir  
End If  
Application.GoTo Reference:="EOutput"  
EOut = Selection.Value
```

```
'Put a backslash on the directory if the user left it off  
If Right(EDir, 1) <> "\" Then  
    EDir = EDir & "\"  
End If
```

```
'Read in the Components workbook file and directory  
Application.GoTo Reference:="CFileName"  
CFile = Selection.Value  
Application.GoTo Reference:="CDirectory"  
CDir = Selection.Value  
Application.GoTo Reference:="CUSPS"  
USPSSheet = Selection.Value  
Application.GoTo Reference:="CPRC"  
PRCSheet = Selection.Value  
If CDir = "" Or CDir = " " Then  
    CDir = ADir  
End If
```

```
Application.GoTo Reference:="COutput"  
COut = Selection.Value  
  
'Put a backslash on the directory if the user left it off  
If Right(CDir, 1) <> "\" Then  
    CDir = CDir & "\"  
End If  
  
'Now read in RF distribution key component numbers and names  
Sheets(RFKeySheet).Activate  
Cells(RFNewComp, RFFirstCol).Select  
ThisComp = Selection.Value  
Cells(RFTitle, RFFirstCol).Select  
ThisName = Selection.Value  
Cells(RFMailVol, RFFirstCol).Select  
ThisMV = Selection.Value  
Response = 0  
If ThisComp = 0 Then  
    Response = MsgBox("You must have at least one component", vbOK)  
    Exit Sub  
End If  
I = 1  
Do While ThisComp > 0  
    RFNewKeys(I) = ThisComp  
    RFTitles(I) = ThisName  
    If ThisMV = "MV" Then  
        RFMV(I) = True  
    Else  
        RFMV(I) = False  
    End If  
    I = I + 1  
    Cells(RFNewComp, RFFirstCol + I - 1).Select  
    ThisComp = Selection.Value  
    Cells(RFTitle, RFFirstCol + I - 1).Select  
    ThisName = Selection.Value  
    Cells(RFMailVol, RFFirstCol + I - 1).Select  
    ThisMV = Selection.Value  
Loop  
  
RFNewKeys(0) = I - 1 'holds the number of new keys  
  
End Sub
```

Sub CheckA()

'This subroutine checks the A Matrix to make sure data does not already exist
'in the components for the roll forward distribution keys.

'The user is presented with a list of components already containing data and
'is asked if s/he wants to continue.

```
Dim RFBadKeys(75) As Long
Dim ThisComp As Long
Dim I, J, iX As Integer
Dim RFNewTot As Integer
Dim BadComp As String
Dim Sep As String
Dim Compl As Long
```

'Check the input matrix and see if any of the components in RFNewComp
'Currently have data in classes 101 - 2000

```
RFNewTot = RFNewKeys(0)
J = 0
Sheets(IMSHEET).Select
For I = 1 To RFNewTot
    Comp = RFNewKeys(I)
    Compl = Comp
    IMRow = 0
    IMCol = 0
    For Class = StartClass To LastClass
        Call IMAAddress(Comp, Class, IMRow, IMCol)
        Cells(IMRow, IMCol).Select
        If Selection.Value <> 0 Then
            J = J + 1
            RFBadKeys(J) = Comp
            Exit For
        End If
    Next Class
Next I
```

'If J > 0 then there is at least one component with existing data.

'Display the components with existing data and ask to exit

```
If J > 0 Then
    BadComp = "The following roll forward components already contain data: "
    For iX = 1 To J
        BadComp = BadComp & Sep & RFBadKeys(iX)
        Sep = ", "
    Next
    BadComp = BadComp & vbCrLf & "Do you want to continue?"
    Response = MsgBox(BadComp, vbYesNo)
    If Response = vbNo Then Exit Sub
    If Response = vbYes Then Response = 0
```

```
End If
```

'OK to continue. Write new keys data into A matrix.

```
End Sub
```

'This subroutine writes out the A Matrix file that has been updated with the
'new roll forward distribution keys.
'Create an array with the values for this distribution key, and then to go the
'InputMatrix and write out the values

Sub WriteA()

```
Dim ThisComp As Long
Dim DKValues(200) As Long
Dim I, J As Long
Dim LastRow As Long
Dim ThisClass As Long
Dim Amount As Double
Dim SaveFile As String
Dim MsgString As String
Dim ThisResponse As Integer

Sheets(RFKeySheet).Activate
ThisBook = ActiveWorkbook.Name

'Find last row of data
Cells(65536, RFClassCol).Select
Selection.End(xlUp).Select
LastRow = Selection.Row

J = 0
Cells(RFNewComp, RFFirstCol + J).Select
ThisComp = Selection.Value

Do While ThisComp > 0
    'Read values into array
    For I = RFFirstClass To LastRow
        Cells(I, RFClassCol).Select
        ThisClass = Selection.Value
        Cells(I, RFFirstCol + J).Select
        DKValues(ThisClass) = Selection.Value
    Next I
    'Write values onto InputMatrix
    Sheets(MenuPage).Select
    'Application.ScreenUpdating = False
    Sheets(IMSHEET).Select
    For I = StartClass To LastClass
        Class = I
        Comp = ThisComp
        Call IMAddress(Comp, Class, IMRow, IMCol)
        Cells(IMRow, IMCol).Select
        Selection.Value = DKValues(I)
    Next I
    'Set up for next component
    J = J + 1
    Sheets(RFKeySheet).Select
    Cells(RFNewComp, RFFirstCol + J).Select
    ThisComp = Selection.Value
```

Loop

```
'Now write out InputMatrix to the A Matrix output file  
Sheets(IMSHEET).Select  
Sheets(IMSHEET).Activate  
Sheets(IMSHEET).Copy
```

```
If (AOut = "" Or AOut = " ") Then AOut = AFile  
SaveFile = ADir & AOut
```

```
'Look to see if a file already exists with output file name
```

```
With Application.FileSearch
```

```
.LookIn = ADir
```

```
.SearchSubFolders = False
```

```
.Filename = AOut
```

```
If .Execute > 0 Then
```

```
MsgString = AOut & " already exists in directory " & ADir & vbCrLf & _  
" Do you want to save anyway?"
```

```
ThisResponse = MsgBox(MsgString, vbYesNoCancel)
```

```
If ThisResponse = 7 Then GoTo End1 'vbNo
```

```
If ThisResponse = 2 Then GoTo End1 'vbCancel
```

```
End If
```

```
End With
```

```
Application.DisplayAlerts = False
```

```
ActiveWorkbook.SaveAs Filename:= _
```

```
SaveFile, FileFormat _
```

```
:=xlNormal, Password:="", WriteResPassword:="", ReadOnlyRecommended:= _
```

```
False, CreateBackup:=False
```

```
Application.DisplayAlerts = True
```

```
Windows(AOut).Close SaveChanges:=False
```

```
'Application.DisplayAlerts = True
```

```
MsgString = AOut & " has been saved in directory " & ADir
```

```
Response = MsgBox(MsgString, vbOKOnly)
```

```
'Windows(ThisBook).Activate
```

```
Sheets(MenuPage).Select
```

```
Exit Sub
```

```
End1:
```

```
Windows(ThisBook).Activate
```

```
Sheets(MenuPage).Select
```

```
Exit Sub
```

```
FileSaveError:
```

```
Eflag = 1
```

```
Resume Next
```

```
End Sub
```

'This subroutine takes the updated roll forward distribution keys and replaces or
'adds them to the Effects worksheet in the specified workbook.
'No longer need to ask if want to write over existing RF keys, because the
'user would have exited earlier if s/he wanted to check prior existing keys

Sub Update_Effects()

```
Dim Eflag As Integer  
Dim ThisComp As Long  
Dim LastRow, FirstRow As Long  
Dim I As Long  
Dim Result As Integer
```

```
Application.StatusBar = "Roll Forward Distribution Key Update: Updating Effects Worksheet in  
Input Workbook"  
ThisBook = ActiveWorkbook.Name  
ThisSheet = ActiveSheet.Name
```

```
Call OpenWB(EFile, EDir, Result)  
Eflag = 0  
On Error GoTo LinkTemplateError
```

'The Effects workbook is now open.

```
If Eflag = 0 Then  
  Sheets(EffectsSheet).Select  
  'The RF only distribution keys are at the bottom of the worksheet.  
  'Go to the bottom, and then search upwards for the first key.  
  Cells(65536, EComp).Select  
  Selection.End(xlUp).Select  
  LastRow = Selection.Row
```

```
I = LastRow  
Cells(I, EComp).Select  
ThisComp = Selection.Value
```

```
Do While ThisComp > LowestRFKey  
  I = I - 1  
  Cells(I, EComp).Select  
  ThisComp = Selection.Value  
Loop
```

```
FirstRow = I + 1
```

```
NumRF = RFNewKeys(0)  
I = FirstRow
```

```
For J = 1 To NumRF  
  Cells(I, EComp).Select  
  Selection.Value = RFNewKeys(J)  
  Cells(I, ECompName).Select  
  Selection.Value = RFTitles(J)  
  Cells(I, EMV).Select  
  If RFMV(J) = True Then  
    Selection.Value = "V"  
  Else
```

```
        Selection.Value = " "
    End If
    I = I + 1
Next J

'Check to see if leftover rows. If so, then clear out data
Do While I <= LastRow
    Cells(I, EComp).Select
    Selection.Value = ""
    Cells(I, ECompName).Select
    Selection.Value = ""
    Cells(I, EMV).Select
    Selection.Value = ""
    I = I + 1
Loop

If (EOut = "" Or EOut = " ") Then EOut = EFile
SaveFile = EDir & EOut
'Look to see if a file already exists with output file name
With Application.FileSearch
    .LookIn = EDir
    .SearchSubFolders = False
    .Filename = EOut
    If .Execute > 0 Then
        MsgString = EOut & " already exists in directory " & EDir & vbCrLf & _
            " Do you want to save anyway?"
        Response = MsgBox(MsgString, vbYesNoCancel)
        If Response = 7 Then GoTo End1 'vbNo
        If Response = 2 Then GoTo End1 'vbCancel
    End If
End With

Application.DisplayAlerts = False
ActiveWorkbook.SaveAs Filename:= _
    SaveFile, FileFormat _
    :=xlNormal, Password:="", WriteResPassword:="", ReadOnlyRecommended:= _
    False, CreateBackup:=False
Application.DisplayAlerts = True
Windows(EOut).Close
MsgString = EOut & " has been saved in directory " & EDir
Response = MsgBox(MsgString, vbOKOnly)
Windows(ThisBook).Activate
Sheets(MenuPage).Select
Exit Sub
End If
End1:

Windows(ThisBook).Activate
Sheets(MenuPage).Select
Application.StatusBar = "Roll Forward Distribution Key Update: Waiting for Next Command"
Exit Sub

LinkTemplateError:
Eflag = 1
Resume Next
End Sub
```

'This subroutine updates the 14xx keys in the Components page with the new
'roll forward distribution keys.

Sub Update_Components(USPS)

```
Dim Eflag As Integer  
Dim ThisComp As Long  
Dim LastRow, FirstRow As Long  
Dim I As Long  
Dim Result As Integer  
Dim CurRows As Long  
Dim DelRows As Long
```

```
ThisBook = ActiveWorkbook.Name  
ThisSheet = ActiveSheet.Name
```

```
Call OpenWB(CFile, CDir, Result)  
Eflag = 0  
On Error GoTo LinkTemplateError
```

'The Components workbook is now open.

```
If Eflag = 0 Then  
  If USPS = True Then  
    Sheets(USPSSheet).Select  
    Application.StatusBar = "Roll Forward Distribution Key Update: Updating USPS Component  
List"  
  ElseIf USPS = False Then  
    Sheets(PRCSheet).Select  
    Application.StatusBar = "Roll Forward Distribution Key Update: Updating PRC Component  
List"
```

```
  End If  
  'Go to the bottom, and then search upwards for the first RF key.  
  Cells(65536, CComp).Select  
  Selection.End(xlUp).Select  
  LastRow = Selection.Row
```

```
  I = LastRow  
  Cells(I, CComp).Select  
  ThisComp = Selection.Value  
  Do Until ThisComp >= LowestRFKey  
    I = I - 1  
    Cells(I, CComp).Select  
    ThisComp = Selection.Value  
  Loop  
  LastRow = I
```

```
  'Now find the first RF key by continuing to search upwards  
  Do While ThisComp > LowestRFKey  
    I = I - 1  
    Cells(I, CComp).Select  
    ThisComp = Selection.Value  
  Loop
```

```
FirstRow = I + 1
```

```
NumRF = RFNewKeys(0)
```

```
I = FirstRow
```

```
'Check to see if need to insert rows to hold distribution key info
```

```
CurRows = LastRow - FirstRow + 1
```

```
Do While CurRows < NumRF
```

```
  Rows(FirstRow).Select
```

```
  Selection.Insert Shift:=xlDown
```

```
  Selection.Font.ColorIndex = 0
```

```
  Selection.Font.Italic = True
```

```
  Selection.Font.Italic = False
```

```
  Selection.Font.Bold = False
```

```
  CurRows = CurRows + 1
```

```
Loop
```

```
For J = 1 To NumRF
```

```
  Cells(I, CComp).Select
```

```
  Selection.Value = RFNewKeys(J)
```

```
  Cells(I, CCompName).Select
```

```
  Selection.Value = RFTitles(J)
```

```
  Cells(I, CLevel).Select
```

```
  Selection.Value = 2
```

```
  Cells(I, CCS).Select
```

```
  Selection.Value = 96
```

```
  Cells(I, CParent).Select
```

```
  Selection.Value = LowestRFKey
```

```
  Cells(I, CSubtotal).Select
```

```
  Selection.Value = "FALSE"
```

```
  Cells(I, CTotOnly).Select
```

```
  Selection.Value = "FALSE"
```

```
  I = I + 1
```

```
Next J
```

```
'Check to see if leftover rows. If so, then delete rows.
```

```
DelRows = LastRow - I + 1
```

```
Do While DelRows > 0
```

```
  Rows(I).Select
```

```
  Selection.Delete Shift:=xlUp
```

```
  DelRows = DelRows - 1
```

```
Loop
```

```
End If
```

```
Application.StatusBar = "Roll Forward Distribution Key Update: Waiting for Next Command"
```

```
Exit Sub
```

```
LinkTemplateError:
```

```
Eflag = 1
```

```
Resume Next
```

```
End Sub
```

Sub WriteComponents()

'Writes out the workbook containing the Components worksheets

```
If (COut = "" Or COut = " ") Then COut = CFile
SaveFile = CDir & COut
'Look to see if a file already exists with output file name
With Application.FileSearch
    .LookIn = CDir
    .SearchSubFolders = False
    .Filename = COut
    If .Execute > 0 Then
        MsgBox = COut & " already exists in directory " & CDir & vbCrLf & _
            " Do you want to save anyway?"
        Response = MsgBox(MsgString, vbYesNoCancel)
        If Response = 7 Then GoTo End1 'vbNo
        If Response = 2 Then GoTo End1 'vbCancel
    End If
End With

Application.DisplayAlerts = False
ActiveWorkbook.SaveAs Filename:= _
    SaveFile, FileFormat _
    :=xlNormal, Password:="", WriteResPassword:="", ReadOnlyRecommended:= _
    False, CreateBackup:=False
Application.DisplayAlerts = True
Windows(COut).Close
MsgString = COut & " has been saved in directory " & CDir
Response = MsgBox(MsgString, vbOKOnly)
'Windows(ThisBook).Activate
Sheets(MenuPage).Select
Exit Sub
'End If

End1:

Windows(ThisBook).Activate
Sheets(MenuPage).Select

End Sub
```

Sub OpenWB(OpenBook, OpenPath, Result As Integer)

```
'New argument, IMode, to be added, but not functioning yet
'Sub OpenWB(OpenBook, OpenPath, Result As Integer, IMode As Integer)

'Input argument OpenBook is the name of the workbook to be opened.
'Input argument OpenPath is the path where the workbook to be opened is located.
'Output argument Result is set as follows:
' 0 = workbook was already open before calling
' 1 = workbook already existed, and is now open
' 2 = workbook had to be created

'Input argument IMode has the following meanings:
' 0 = create the workbook if it does not exist
' 1 = open it if it's not open, but do not create it if it does not exist
' 2 = activate it if it is already open, but do not open it
'In all cases, the workbook will be open and will be the active workbook upon exiting this sub

'Fill holds the next available file number
'EFlag is an error flag
Dim Fill, Eflag As Integer

'FullBookName is the concatenated path and workbook name
Dim FullBookName As String

'OldStatusText is the text that was displayed in the statusbar before coming here
Dim OldStatusText As String

Eflag = 0
Result = 0
FullBookName = OpenPath & OpenBook

'Check to see whether the file needs to be created
'Try to open the workbook. This generates an error (which is trapped) if it doesn't exist
'This uses a DOS-style Open statement rather than an Excel (Workbooks.Open) to test whether
'the file exists. FreeFile is a function that returns the next available file number.
'OldStatusText = Application.StatusBar
'Application.StatusBar = CurYear & ": Checking for file " & FullBookName
Fill = FreeFile
On Error GoTo OpenWBErr
Open FullBookName For Input As Fill
Close Fill
On Error GoTo 0
If Eflag > 0 Then
    Workbooks.Add
    ActiveWorkbook.SaveAs Filename:=FullBookName, _
        FileFormat:=xlNormal, _
        Password:="", _
        WriteResPassword:="", _
        ReadOnlyRecommended:=False, _
        CreateBackup:=False

    Result = 2
    'Application.StatusBar = OldStatusText
Exit Sub
End If
```

```
'If we get here, the file exists. Check to see if it is already open
'This will generate Error 9 (subscript out of range) if the workbook is not open
'Note that this routine will not distinguish between the desired workbook and
'one with the same name from a different folder that is already open.
'If there is another workbook with the same name already open, this routine will
'think that the desired workbook is open already.
On Error GoTo OpenWBErr
Windows(OpenBook).Activate
On Error GoTo 0
If Eflag > 0 Then
    'Application.StatusBar = CurYear & ": Opening " & FullBookName
'The second paramter of the Open method (False) tells Excel NOT TO UPDATE LINKS
'Of course, there shouldn't be any links in these workbooks.
    Workbooks.Open FullBookName, False
    Result = 1
End If

Exit Sub

OpenWBErr:
Eflag = 1
Resume Next

End Sub
```

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
AddendNum	Location of this column on Input Table for cost reductions / other programs	Integer	ModPublicVars	ReadFileNames	DKInit	Input Table worksheet DistKeys
AllWorkbookNames (MaxYears, MaxEffects)	Names of workbooks that could be used for all years and all effects	String array	ModPublicVars	InitializeWorkbookNames	FileCheck	
BaseYear	Base Year from RFModel Menu page	Integer	ModPublicVars	Init	NOT USED	RFModel worksheet RFMenuPage
BookName	workbook name without the path	String	ModPublicVars	NOT USED	NOT USED	
ClassColumn	ClassColumn is the column containing the mail class number	Integer	ModPublicVars	Init	Piggybacks GetRatios WriteIndFormula CROPWorkbook Type1 - Type 4 CROPSumtoCS InitClassRow	
Classes(MaxClass)	Contains data from MailClass worksheet in RF Model	MailClassData array	ModPublicVars	ClassArrayInit	ClassSubtotals TemplateFormulas NVAWFormulas GetClasses MakeAdjVolFormulas Piggybacks WriteIndFormula Type1 - Type4 CROPSumtoCS	RFModel worksheet MailClasses
ClassFormulas	Holds the formulas to calculate mail class subtotals	String array	ModPublicVars	ClassSubtotals	MakeAdjVolFormulas Piggybacks WriteIndFormula Type1 - Type 4 CROPSumtoCS TemplateFormulas NVAWFormulas	
ClassMax	Maximum number of entries in Mail Class array	Long	ModPublicVars	ClassArrayInit	GetClasses	RFModel worksheet MailClasses
ClassOffset	ClassOffset is the row number where mail class costs begin on each page	Integer		Init	Piggybacks GetRatios WriteIndFormula CROPWorkbook Type1 - Type 4 InitClassRow ComponentSubtotals	
ClassRow(MaxClass)	Row for each class on the output worksheets	Long array	ModPublicVars	InitClassRow	ClassSubtotals ComponentSubtotals	
CloseBooks(100)	Closebooks contains a list of workbooks that need to be (saved and) closed at the end	String array	ModPublicVars	NOT USED	NOT USED	
Cmatrix	Column on input matrix	Long	ModPublicVars	NOT USED	NOT USED	
COffSet	Number of columns by which components are offset in the input matrix	Integer	ModPublicVars	Init	Init NVAWFormulas RefreshInputMatrix MakeOutputMatrix MakeAdjVolFormulas Piggybacks WriteIndFormula CROPWorkbook Type1 - Type4	RFModel worksheet BaseYear

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
Col1	Starting column for labels in effect workbooks	Long	ModPublicVars	Init	NVAWFormulas GetClasses MakeOutputMatrix MakeAdjVolFormulas MAEffect TemplateFormulas	
ColHeaders ()	Labels for each column of the Input Matrix shee	String array	ModPublicVars		NOT USED	
ComponentCount	Number of entries in Components array	Integer	ModPublicVars	Init	NVAWFormulas AddComps FindChildren CROPSumtoCS	RFModel Components worksheet
ComponentIndex	Index into Components array	Long	ModPublicVars	Init	ComponentSubtotals	RFModel Components worksheet
ComponentName	Name of this component	String	ModPublicVars	Init	MakeAdjVolFormulas Piggybacks GetRatios	RFModel Components worksheet
ComponentOffset	ComponentOffset is the column number where component costs begin on each page	Integer	ModPublicVars	Init	Piggybacks GetRatios WriteIndFormula Type1 - Type4 CROPSumtoCS ComponentSubtotals AddComps MakeAdjVolFormulas	
ComponentRow	ComponentRow is the row with the component number for each column in the effect workbooks	Integer	ModPublicVars	Init	Piggybacks GetRatios WriteIndFormula CROPWorkbook Type1 - Type4 CROPSumtoCS ComponentSubtotals TemplateFormulas NVAWFormulas AddComps MakeAdjVolFormulas	
Components(MaxCompon)	Contains data from Components worksheet in RFModel	ComponentType array	ModPublicVars	Init	CROPSumtoCS ComponentSubtotals	RFModel Components worksheet
ComponentSheet	Name of worksheet in RFModel containing components	String	ModPublicVars	Init	MAEffect Init AddCSSheet	
CROPAmt	Column in CostReduction/Other Program input table worksheet	Integer	ModPublicVars	CROPInit, Type2, Type4	Type2, Type4	RFModel EffectsParameters worksheet
CROPComp	Column in CostReduction/Other Program input table worksheet	Integer	ModPublicVars	CROPInit, Type2, Type4	Type2, Type4	RFModel EffectsParameters worksheet
CROPCompName	Column in CostReduction/Other Program input table worksheet	Integer	ModPublicVars	CROPInit, Type2, Type4	Type2, Type4	RFModel EffectsParameters worksheet
CROPDefs(MaxCROP)	Input data from Input Table worksheet for cost reductions / other programs	CROPData array	ModPublicVars	CROPInit	CROPInit CROPWorkbook CROPDist Type1 - Type4 CROPSumtoCS	Input Table worksheet CostReductions / Other Programs
CROPDK	Column in CostReduction/Other Program input table worksheet	Integer	ModPublicVars	CROPInit, Type2, Type4	Type2, Type4	RFModel EffectsParameters worksheet

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
CROPName	Column in CostReduction/Other Program input table worksheet	Integer	ModPublicVars	CROPInit, Type2, Type4	Type2, Type4	RFModel EffectsParameters worksheet
CROPSubTot	Column in CostReduction/Other Program input table worksheet	Integer	ModPublicVars	CROPInit, Type2, Type4	Type2, Type4	RFModel EffectsParameters worksheet
CSName(MaxCS)	Name for each cost segment	String array	ModPublicVars	CSNameInit	AddCSSheet MAEffect	RFModel worksheet CostSegments
CSSheet	The name of the current worksheet, used in GetClasses to determine the correct labels to add to the worksheet	String	ModPublicVars NVAWFormulas (local) MakeOutputMatrix (local) MakeAdjVolFormulas (local) TemplateFormulas (local)	AddCSSheet NVAWFormulas (local) MakeOutputMatrix (local) MakeAdjVolFormulas (local) TemplateFormulas (local)	GetClasses NVAWFormulas (local) MakeOutputMatrix (local) MakeAdjVolFormulas (local) TemplateFormulas (local)	
CSSheetMode	Indicator for how to build the Cost Segment worksheets. Value of 'VolAdj' means add a column with volume adjustment factors	String	ModPublicVars	VEffect CROPWorkbook MAEffect Meffect	GetClasses MakeOutputMatrix	
Ctype	Not Used		ModPublicVars	Driver	NOT USED	
CurYear	Current Year - from Base Year to Test Year	Integer	ModPublicVars	NOT USED	NOT USED	
DefaultDirectory	Default directory to find workbooks	String	ModPublicVars	ReadFileNames	ReadFileNames FileCheck Driver ErrorCheckDriver OpenWB VEffect MAEffect	RFModel worksheet Scenarios
DistKeysSheet	Name of the Distribution Keys worksheet	String	ModPublicVars	ReadFileNames	DKInit	RFModel worksheet EffectsParameters
DistributionKey(MaxComps)	The distribution key number for each indirect component	Integer array	ModPublicVars	IndirectInit	Piggybacks	Input Table worksheet EffectList
DKDefinitions(MaxDefs)	Contains Mail Class information from Input Table worksheet DistKeys	DistKeyData array	ModPublicVars	DKInit	Piggybacks	Input Table worksheet DistKeys
DKNum	Location of this column on Input Table for cost reductions / other programs	Integer	ModPublicVars	ReadFileNames	DKInit IndirectInit	RFModel worksheet EffectsParameters
ECompName	Column component name column on EffectList sheet in Input Table	Integer	ModPublicVars	ReadFileNames	DKInit NVAWFormulas MakeAdjVolFormulas TemplateFormulas	RFModel worksheet EffectsParameters
ECompon	Column for component number column on EffectList sheet in Input Table	Integer	ModPublicVars	ReadFileNames	NVAWFormulas MakeAdjVolFormulas TemplateFormulas chkEffectList	RFModel worksheet EffectsParameters

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
EffectData(MaxEffects)	From RFModel worksheet EffectParameters, describing where to find input data	EffectInfo array	ModPublicVars	ReadFileNames	IndirectInit MakeBookNames InitializeWorkbookNames Driver MEffect NVAWFormulas VEffect MakeAdjVolFormulas CROPInit CROPWorkbook Type2, Type4 MAEffect TemplateFormulas ErrorCheckDriver chkEffectList AddCSSheet FindPreviousEffect	RFModel worksheet EffectsParameters
EffectLabel	Label for current effect	String	ModPublicVars	MEffect NVAWFormulas VEffect CROPWorkbook MAEffect	MEffect NVAWFormulas AddCSSheet VEffect Piggybacks CROPWorkbook MAEffect	RFModel worksheet EffectsParameters
EffectsSheet	Name of the Effects Sheet	String	ModPublicVars	Driver	NVAWFormulas MakeAdjVolFormulas TemplateFormulas IndirectInit	RFModel worksheet EffectsParameters
EFlag	Error indicator	Integer	modPublicVars	Driver RefreshInputMatrix MakeOutputMatrix VEffect CROPSumtoCS MAEffect MEffect MEffect ReadFileNames FileCheck GetInputMatrix Piggybacks (local) CROPWorkbook (local) OpenWB (local) AddCSSheet (local)	RefreshInputMatrix MakeOutputMatrix VEffect CROPSumtoCS MAEffect MEffect FileCheck GetInputMatrix Piggybacks (local) CROPWorkbook (local) OpenWB (local) AddCSSheet (local)	
ErrorCount	The number of errors that were found in the error checking routine.	Long	ModPublicVars	ErrorCheckDriver chkEffectList	Driver chkEffectList	
FileLoc	Full file name including path	String	modPublicVars	NOT USED	NOT USED	
Fill	Next available file number	Integer	modPublicVars, OpenWB (local)	NOT USED OpenWB (local)	NOT USED OpenWB (local)	
FYDirectory	Directory to find effect workbooks	String	modPublicVars	NOT USED	CROPWorkbook Type2, Type4 MAEffect MEffect	

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
FYIMatrix	Name of the base year Input Matrix sheet in the RFModel workbook	String	modPublicVars	ReadFileNames	VEffect MEffect CROPWorkbook GetInputMatrix	RFModel worksheet BaseYear
FYWorkbook	Name of the first year Cost Level workbook	String	modPublicVars	NOT USED	NOT USED	
iEffect	Current effect	Integer	modPublicVars	Driver	Driver IndirectInit GetInputMatrix MEffect NVAWFormulas VEffect MakeAdjVolFormulas CROPInit CROPWorkbook Type2, Type4 MAEffect TemplateFormulas ErrorCheckDriver chkEffectList AddCSSheet FindPreviousEffect	
iEndEffect	End number of effect within a single year run	Integer	modPublicVars	ReadFileNames CheckEffectOrder	Driver ReadFileNames FileCheck ErrorCheckDriver	RFModel workbook RFMenuPage
iEndYear	Index of the year (in a multi-year run) to end with	Integer	modPublicVars	ReadFileNames CheckYearOrder	Driver ErrorCheckDriver FileCheck ReadFileNames CheckYearOrder	RFModel workbook RFMenuPage
IMatrixBook	Name of workbook containing input cost matrix	String	modPublicVars	MakeBookNames Driver ReadFileNames	GetInputMatrix RefreshInputMatrix	RFModel worksheet BaseYear
IMatrixCheck	Whether to perform data check on base year Input Matrix	Boolean	modPublicVars	ReadFileNames	RefreshInputMatrix	
IMatrixDir	Directory to find 200 x 1600 cost matrix	String	modPublicVars	Driver ReadFileNames	GetInputMatrix RefreshInputMatrix ReadFileNames	RFModel worksheet BaseYear
IMatrixSheet	Name of sheet containing the cost matrix		modPublicVars	ReadFileNames CROPWorkbook GetInputMatrix	RefreshInputMatrix MakeOutputMatrix GetInputMatrix CROPWorkbook MAEffect	RFModel worksheet BaseYear
IMatrixStart	Name of the worksheet in the RFModel workbook containing the base year Input Matrix	String	modPublicVars	ReadFileNames	RefreshInputMatrix GetInputMatrix	RFModel worksheet BaseYear
IndirectFactor(MaxComps)	Contains a 1 for each component needing an indirect cost calculator	Integer array	modPublicVars	IndirectInit	Piggybacks	Input Table worksheet Effect list
InputClassStart	The first class (column) of the input matrix that needs to be read. All classes less than this will be set to zero	Integer	modPublicVars	Init	MakeOutputMatrix	RFModel worksheet Limits

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
InputDirectory	Directory containing the Input Table workbook	String	modPublicVars	ReadFileNames	DKInit IndirectInit NVAWFormulas VEffect CROPInit Type2, Type4 TemplateFormulas	RFModel worksheet Scenarios
InputEffect	Effect whose Output Matrix serves as input to this effect	Integer	modPublicVars	FindPreviousEffect	MEffect VEffect CROPWorkbook	RFModel worksheet EffectsParameters
InputTablesFile(10)	Name of the Input Table file for up to 10 years including BR and AR for test year	String array	modPublicVars	ReadFileNames	FileCheck MakeBookNames ErrorCheckDriver	RFModel worksheet InputWorkbooks
InputWorkbook	Name of the Input Table workbook	String	modPublicVars	MakeBookNames	DKInit IndirectInit NVAWFormulas VEffect MakeAdjVolFormulas CROPInit Type2, Type4 TemplateFormulas	RFModel worksheet InputWorkbooks
iStartEffect	Start number of effect within a single year run	Integer	modPublicVars	CheckEffectOrder ReadFileNames	Driver ErrorCheckDriver ReadFileNames FileCheck	RFModel worksheet RFMenuPage
iStartYear	Index of the year (in a multi-year run) to start with	Integer	modPublicVars	ReadFileNames CheckYearOrder	FileCheck Driver ErrorCheckDriver ReadFileNames	RFModel worksheet RFMenuPage
iYear	Index into YearLabel matrix	Integer	modPublicVars	ReadFileNames	Driver ErrorCheckDriver MakeBookNames GetInputMatrix FileCheck	RFModel worksheet RFMenuPage
LastBackSlash	Position of the last backslash in FileLoc	Integer	modPublicVars	NOT USED	NOT USED	
LastCostCS	The last cost segment containing cost component data	Integer	modPublicVars	Init	AddComps CROPSumtoCS ComponentSubtotals	RFModel worksheet Limits
MainBook	Name of the RFModel workbook (the workbook from which the buttons were clicked)	String	modPublicVars	Init	Init ReadFileNames GetInputMatrix Driver Cleanup ClassArrayInit AddCSSheet MAEffect	
MainSheet	Sheet in MainBook that contains the button that was clicked	String	modPublicVars	Init	Driver Init	
Max2Comp	The maximum number of components to spread amount over (Type 2)	Integer	modPublicVars	Init	Type2, Type4	RFModel worksheet Limits

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
MaxClass	Size of the array that holds Mail Class data	Integer	modPublicVars	Init	NVAWFormulas Init GetClasses MakeAdjVolFormulas Piggybacks GetRatios WriteIndFormula Type1 - Type4 CROPSumtoCS ClassSubtotals InitClassRow TemplateFormulas	RFModel worksheet Limits
MaxCompon	Maximum number of Components	Integer	modPublicVars	Init	ComponentSubtotals Init	RFModel worksheet Limits
MaxComps	The maximum number of components in the InputMatrix	Long	modPublicVars	Init	Init IndirectInit Piggybacks	RFModel worksheet Limits
MaxCROP	The maximum number of cost reductions or other programs lines in the input data file	Integer	modPublicVars	Init	Init CROPInit	RFModel worksheet Limits
MaxCS	Maximum number of Cost Segments that are allowed	Integer	modPublicVars	Init	Init CSNameInit AddCSSheet MAEffect	RFModel worksheet Limits
MaxDefs	The maximum number of distribution key definitions in the input file for distribution keys	Integer	modPublicVars	Init	Init	RFModel worksheet Limits
MaxDKComp	The most addends in a distribution key	Integer	modPublicVars	Init	Piggybacks	RFModel worksheet Limits
MaxEffects	Maximum number of effects	Integer	modPublicVars	Init	Init	RFModel worksheet Limits
MaxYears	Maximum number of years that can be analyzed in one ru	Integer	modPublicVars	Init	Init	RFModel worksheet Limits
Msg	Used for message boxes	String	modPublicVars	GetInputMatrix Driver RefreshInputMatrix ReadFileNames (local)	GetInputMatrix Driver RefreshInputMatrix ReadFileNames (local)	
MsgDisplay	Controls whether to display a message when Scenario List has been updated	Boolean	modPublicVars	Auto_Open UpdateLists UpdateEffectsLists UpdateYearsLists	UpdateScenarioLists UpdateEffectsLists UpdateYearsLists	
NameLen	Length of FileLoc	Integer	modPublicVars	NOT USED	NOT USED	
NumEffects	Number of total effects in a year	Integer	modPublicVars	ReadFileNames	FileCheck ReadFileNames MakeBookNames Driver ErrorCheckDriver AddCSSheet FindPreviousEffect	RFModel worksheet EffectsParameters
OtherClass	The class number for other (institutional) cos	Integer	modPublicVars	Init	WriteIndFormula	RFModel worksheet Limits
PathName	Path name without the book name	String	modPublicVars	NOT USED	NOT USED	
PrevYearLabel (10)	Label of the year that serves as input to this one	String	modPublicVars	ReadFileNames	Driver	RFModel worksheet InputWorkbooks
RatioVector(MaxClass)	The percentage change in cost after a effect is applied to the direct components comprising the indirect component's distribution key	Double array	modPublicVars	Piggybacks	NOT USED	
ReverseFileName	Not used?	String	modPublicVars	NOT USED	NOT USED	
RMatrix	Row on input matrix	Long	modPublicVars	NOT USED	NOT USED	

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
ROffSet	Number of rows by which components are offset in the input matrix	Integer	modPublicVars	Init	Init NVAWFormulas RefreshInputMatrix MakeOutputMatrix MakeAdjVolFormulas Piggybacks WriteIndFormulas TemplateFormulas Type1 - Type4	RFModel worksheet BaseYear
Row1	Starting row for labels in effect workbooks	Long	modPublicVars	Init	NVAWFormulas GetClasses MakeOutputMatrix MakeAdjVolFormulas MAEffect TemplateFormulas Init	
RowHeaders ()	Labels for each row of the Input Matrix sheet	String array	modPublicVars	Init	NOT USED	
SaveCloseNames(100)	List of workbooks that need to be closed at the end of the run	String array	modPublicVars	Init OpenWB	SaveCloseWorkbooks	
SCCcount	Number of workbooks in the array	Integer	modPublicVars	Init OpenWB	Init SaveCloseWorkbooks	
ScenarioRow	Row on Scenarios worksheet for selected scenario	Long	modPublicVars	Init ScenarioChange	ReadFileNames	RFModel worksheet RFMenuPage
ScenDescription	Description of scenario	String	modPublicVars	ReadFileNames	ReadFileNames AddCSSheet Piggybacks CROPWorkbook MAEffect	RFModel worksheet Scenarios
ScenName	Name of the scenario	String	modPublicVars	ReadFileNames	NOT USED	RFModel worksheet Scenarios
ScenWB	Scenario component of workbook name	String	modPublicVars	ReadFileNames	MakeBookNames InitializeWorkbookNames	RFModel worksheet Scenarios
SegCompon(9999)	Array to tell which Cost Segment each Component belongs to.	Byte	modPublicVars	Init	NVAWFormulas MakeAdjVolFormulas Piggybacks WriteIndFormula TemplateFormulas	RFModel Components worksheet
SheetName	Sheet to copy the cost matrix from	String	modPublicVars AddComps (local) AddCSSheet (local) Piggybacks (local) GetRatios (local) WriteIndFormula (local) CROPWorkbook (local) Type3 (local) Type2 (local) CROPsumtoCS (local) ClassSubtotals (local) ComponentSubtotals (local)	MAEffect AddComps (local) AddCSSheet (local) Piggybacks (local) GetRatios (local) WriteIndFormula (local) CROPWorkbook (local) Type3 (local) Type2 (local) CROPsumtoCS (local) NOT USED ComponentSubtotals (local)	MAEffect AddComps (local) AddCSSheet (local) Piggybacks (local) GetRatios (local) WriteIndFormula (local) CROPWorkbook (local) Type3 (local) Type2 (local) CROPsumtoCS (local) NOT USED ComponentSubtotals (local)	

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
StatusLabel	Status text	String	modPublicVars	VEffect CROPWorkbook MAEffect MEffect NVAWFormulas	VEffect ComponentSubtotals MAEffect MEffect NVAWFormulas MakeOutputMatrix	
strCurYear	String representation of the current year	String	modPublicVars	Driver	MAEffect TemplateFormulas MakeBookNames MEffect NVAWFormulas OpenWB AddCSSheet VEffect MakeAdjVolFormulas Piggybacks CROPWorkbook	
strErrorCheck ()	each element of which contains a description of an input error that was found	String array	modPublicVars	chkEffectList	Driver	
Table5Sheet	Name of the Cost Reductions workshee	String	modPublicVars	NOT USED	NOT USED	
Table6Sheet	Name of the Other Programs workshee	String	modPublicVars	NOT USED	NOT USED	
Table7Sheet	Name of the Workyear Mix workshee	String	modPublicVars	NOT USED	NOT USED	
TestYear	Test Year for analysis	Integer	modPublicVars	Init	NOT USED	RFModel worksheet RFMenuPage
ThisBook	Name of the the current workbook	String	modPublicVars	Init RefreshInputMatrix MAEffect	Driver Init RefreshInputMatrix	
ThisSheet	Name of the current worksheet	String	modPublicVars	Init RefreshInputMatrix MAEffect	Init RefreshInputMatrix	
TotalEffects	Total effects available for run whether selected or not	Integer	modPublicVars	UpdateEffectsList	InitializeWorkbookNames	RFModel worksheet EffectsParameters
TotalYears	Total years available for run whether selected or not	Integer	modPublicVars	UpdateYearsList	InitializeWorkbookNames	RFModel worksheet InputWorkbooks
TotClass	The class number for total accrued cost	Integer	modPublicVars	Init	WriteIndFormula Type1 - Type4 ComponentSubtotals	RFModel worksheet Limits
TotVVClass	The class number for total attributable (volume variable) cost	Integer	modPublicVars	Init	WriteIndFormula MakeAdjVolFormulas	RFModel worksheet Limits
UsedInRF	Location of this column on Input Table worksheet for cost reductions other programs	Integer	modPublicVars	ReadFileNames	NOT USED	RFModel worksheet EffectsParameters
USPSFlag	1 indicates USPS analysis, 2 indicates PRC analysis	Integer	modPublicVars	Init	Init	RFModel worksheet RFMenuPage
VClassCol	Column number for Volume Adjustment class on VolAdj worksheet	Integer	modPublicVars	ReadFileNames	VEffect	RFModel worksheet EffectsParameters
VFactorCol	Column number for Volume Adjustment factors on VolAdj worksheet	Integer	modPublicVars	ReadFileNames	VEffect	RFModel worksheet EffectsParameters
VolAdjClass(MaxClass)	Holds volume adjustment factors	Double, array	modPublicVars	VEffect Init	GetClasses	Input Table worksheet VolAdj
VolAdjFactor(MaxClass)	Holds mail class corresponding to volume adjustment factors	Double, array	modPublicVars	VEffect Init	GetClasses	Input Table worksheet VolAdj
VolAdjSheet	Name of the Volume Adjustment factors sheet	String	modPublicVars	ReadFileNames	VEffect	RFModel worksheet EffectsParameters

XII. Directory of Global Variables Used in Roll Forward Model

Public Variable Name	Description	Data Type	Declared	Where set	Where used	Data Source (If Applicable)
YearLabel(10)	Array holding labels (strings) of the years to be analyzed	String array	modPublicVars	ReadFileNames	InitializeWorkbookNames Driver	RFModel worksheet InputWorkbooks
ynCreateWB	Indicates whether the user wants to create workbooks as needed, or overwrite existing workbooks	Boolean	modPublicVars	Init	FileCheck Driver	RFModel worksheet RFMenuPage
ynMultiYear	True if user wants to do a multi-year run, False if user wants to run selected effects in a single year	Boolean	modPublicVars	MultiYearOptionClick SingleYearOptionClick Init	Driver ErrorCheckDriver ReadFileNames FileCheck	RFModel worksheet RFMenuPage
ynSaveClose	Indicates whether the user wants to save and close workbooks at the end of the run	Boolean	modPublicVars	Init	SaveCloseWorkbooks	RFModel worksheet RFMenuPage

Definition of User Defined Data Types

Data Type Definition	Description of Elements
Public Type DistKeyData DKNumber As Integer DKOrder As Integer DKAddends As Integer DKComponents(150) As Integer End Type	Data type to hold input data from Table worksheet for distribution key Distribution Key Number Distribution Key Order Distribution Key Number of Addends Distribution Key Addends (up to 150 allowed)
Public Type CROPData COName As String COAmount As Double COComp As Long COCompName As String COType As Integer CODK As Double CODKType As String End Type	Data type to hold input data from Table worksheet for cost reductions / other program Name of cost reduction or other program Amount of cost reduction / other program Component number cost reduction / other program applied Name of component cost reduction / other program applied Type (1, 2, 3) Distribution key for amount (type 3), or component to spread amount to (type 3) If component to spread amount to is on the Components List (type 3)
Public Type EffectInfo EffectLabel As String EffectName As String EffectType As String InputPage As String InputCol As Integer NameCol As Integer AmtCol As Integer CompCol As Integer CompNameCol As Integer DKCol As Integer DKTypeCol As Integer InputMatrix As String WorkbookName As String End Type	Data type to hold information about effects, including where to find input data The label that will be part of the file name Name of this effect Type of effect (ie. A, MA, V, M) Name of the worksheet in the Table file that contains the input data for this effect Column number where input data resides on Table worksheet Column number where name of cost reduction / other program resides on Table worksheet Column number where Amount data resides on Table worksheet Column number where Component data resides on Table worksheet Column number where Component Name data resides on Table worksheet Column number where Distribution Key data resides on Table worksheet Column number where Distribution Key Type resides on Table worksheet The label of the effect that serves as the input to this effect Name of the workbook that is the output for this effect
Public Type MailClassData ClassName As String Component As Integer Level As Integer Parent As Integer SubTot As Boolean TotOnly As Boolean Header As Boolean Sort As String Display As Integer End Type	Data type to hold mail class data from MailClass worksheet in RFMoc Class Name Class Number Level (parent/child relationship) Parent class for this class Whether class is a subtotal class Not Used Not Used Not Used Indicates how to display this class on the output worksheet

Definition of User Defined Data Types

Data Type Definition	Description of Elements
Public Type ComponentType CompNumber As Long CompName As String CompLevel As Integer CompCS As Integer CompParent As Long CompSubTot As Boolean CompTotOnly As Boolean CompHeader As Boolean CompSort As String End Type	Data type to hold data component from Components worksheet in RFMO Component Number Component Name Level (Parent/Child relationship) Cost Segment Parent of this component Whether this component is a subtotal Not used Not used Not used
Public Type CompDist WS As String Col As Long End Type	Data type to hold locations of distributed cost reductions / other programs for a component Worksheet where distributed amount is located Column on worksheet where distributed amount is located