Arius®

API User Guide

An overview of how to use the exposed Arius functions within Microsoft Excel VBA

Milliman

IT TAKES VISION

Information in this document is subject to change without notice. The software described in this manual is furnished under a license agreement. The software may be used or copied only in accordance with the terms of that agreement.

No portion of the contents of this publication may be reproduced or transmitted in any form or by any means without the express written permission of Milliman.

Milliman, Inc.
3424 Peachtree Road NE, Suite 1900
Atlanta GA 30326 USA

Tel +1 800 404 2276
Fax +1 404 237 6984

ActuarialSoftware.com

© Copyright 2006-2021 Milliman, Inc. All Rights Reserved.
This document is the proprietary and confidential property of Milliman, Inc.
Arius® is a registered trademark of Milliman, Inc. All other trademarks are property of their respective owners.

Table of Contents

1. General information

The Arius Application Program Interface (API) extends Arius' capabilities by exposing key functions that allow you to automate the exchange of data between Arius and other tools such as Microsoft Excel and Microsoft's Visual Basic for Applications (VBA). Example uses of the API with Excel include

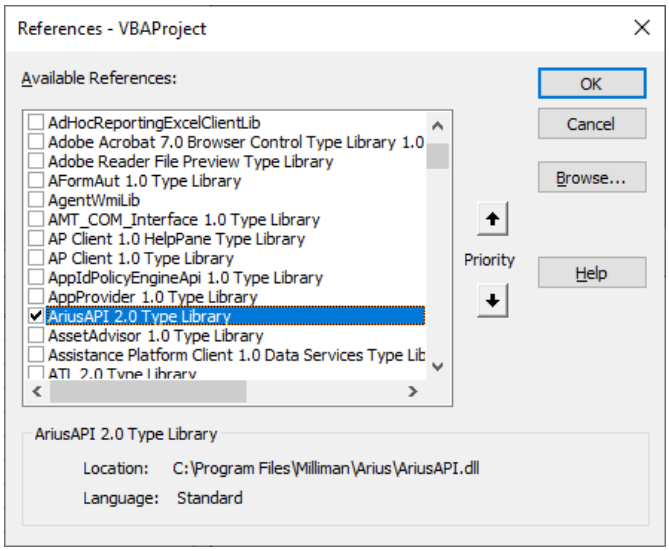
- spreadsheets that import data from Arius, conduct additional analyses or implement their own workflow, and send results back to Arius; and
- extending the reporting capabilities of Arius by summarizing results from multiple Arius files into one spreadsheet.

The Arius API focuses on exchanging data with Arius (*.apj) files.

Most users will use the API to extend Arius' capabilities and integrate with Excel. However, the API can be used with other tools as well, such as other Microsoft Office products using VBA or with other programming tools such as Microsoft Visual Basic, C# or C++. It is beyond the scope of this document to instruct how to use the Arius COM Interop API in environments outside of Excel and VBA, and we do not provide any examples other than Visual Basic and Excel, but we would be happy to provide additional technical assistance to help you implement the API in other technical programming environments.

2. Before you get started with the API

In order to use the Arius API in your workbooks or the examples provided, you will need to make certain that there is an Excel VBA Project reference to the Arius API 2.0 Type Library. All of the examples provided should already include this reference. The location of the library will change depending on where you have installed Arius on your computer.



The Arius API was designed to work with Excel version 2010 and later (as well as VBA).

You will need to add the reference in Excel to the Arius API library. For example, in Excel 2016, press Alt+F11 to open the VBA Editor (or go to DEVELOPER | VISUAL BASIC). Then select TOOLS | REFERENCES… and browse to find AriusAPI.dll (see the example location in the picture above for the most typical install location).

Each of the Excel examples provided with the system include macros and functions that either wrap the Arius API or provide additional Excel function calls to build templates and insert formulas. You must enable Excel macros when you open each example workbook in order for them to function.

If you do not have automatic recalculation in Excel enabled, you will need to press F9 to force the functions and macros on the Excel spreadsheet to execute.

3. Example Excel workbooks

Your Arius installation provides several example workbooks demonstrating how to use the API from within Excel. Each of these examples stands alone and demonstrates techniques that may help you in customizing your own reports, exhibits, and spreadsheets.

All examples are located under the Documents\Milliman\Arius\API directory in a folder titled ExcelVBA Samples. Each example is contained in its own subfolder. You can use these examples exactly as they are, or you may customize them to best fit your needs. Milliman consultants are also available to assist you in creating custom solutions, as well as in supporting the general use of these APIs.

We recommend that you create a copy of the original folder prior to editing the examples, in order to preserve the original form and function of the sample files. Should you accidentally delete or edit the original example folders, reinstalling the software will return the originals to your computer.

The examples introduce only a few functions and perform basic tasks; they were designed to demonstrate the following capabilities:

- Provide a button that will either populate or modify the worksheet with function and API calls.
- Use Excel to execute functions on the worksheet which exchange data with Arius.

All of the examples were constructed specifically to work with the example data provided. A few of the samples perform basic operations that are generic with respect to the data, and they will work regardless of the size, number, and dimension of the Arius sample files. Others adhere to more specific formatting requirements, and you will need to manually resize the output areas to accommodate the size of the data being retrieved—for example, Example2 will require size adjustments for the output range area to account for the number or rows and columns in the file to properly display the Exhibit triangle and Method column. Where possible, the examples use array formulas, so increasing or decreasing the output range area (by inserting or deleting rows and columns) is often all that is necessary to accommodate different file sizes.

To make the APIs easier to use, some of the Excel examples were built with a few VBA functions that wrap several of the API calls together (as well other native Excel VBA functions to help with formatting). These Visual Basic functions may be less difficult to call from within Excel, and they allow you to perform operations such as declaring and instantiating the Arius automation object, which, in turn, starts an instance of the Arius application object. Using an Arius instance, other Arius APIs are able to make API calls from inside the function. Normally, the examples instantiate an instance of the Arius automation object when the workbook is loaded. It is declared as a global variable so other functions have access to it. As you study the code, you will see how the Excel VBA functions encapsulate multiple Arius API calls (for example, start Arius, open an Arius file, get the file properties, close the Arius file, etc.) and embed native Excel function calls in order to format the output. This is analogous to writing macros in Excel that perform several functions, so that you only need to call the single macro.

Alternatively, two examples (example 4b which gets and sends multiple data input tables from/to Arius, and example 11 which gets multiple data Input tables, Exhibits, Methods and Reports from Arius) have been provided which avoid the use of these Excel VBA functions, to the extent possible, to enhance readability of the code. These examples populate the workbook with data values rather than Excel VBA defined functions. For those who are new to VBA and the Arius API, the code in these examples may be easier to understand. Example 4B in particular includes a VBA and API tutorial for

NOTE:

Example 4b is designed specifically for those who are new to VBA and the Arius API.

those who are new to programming or working with VBA within Excel. These examples, like all of the examples, can be used exactly as they are with no modification.

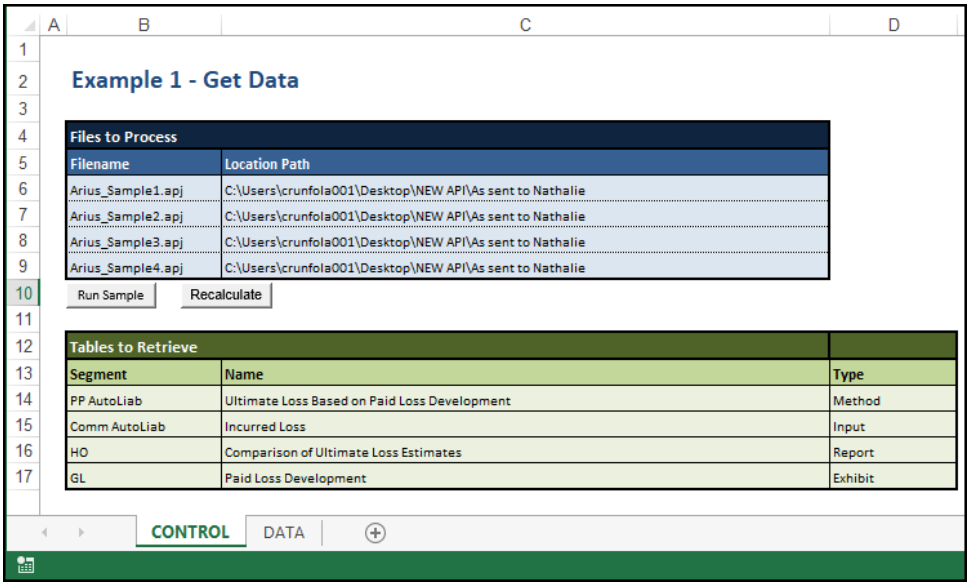
To see the example user defined VBA functions which encapsulate the Arius API calls, do the following:

1. Open any of the Arius example workbooks.
2. Make sure you Enable Macros when the workbook opens.
3. Right-click on any worksheet tab in the workbook.
4. Select **View Code**.
5. Double-click one or more of the modules listed on the tree inside the Excel VBA editor.

You should experiment with the code by inserting break points, changing values and observing the results in the Excel sample workbook. You are encouraged to modify the code and experiment. However, we strongly recommend you make a copy of the workbook first so you can always refer back to the original (just in case).

Following is a description of each example installed with the Arius automation module.

EXAMPLE 1: BATCH GET DATA



Example 1 - Get Data

Files to Process	
Filename	Location Path
Arius_Sample1.apj	C:\Users\crunfola001\Desktop\NEW API\As sent to Nathalie
Arius_Sample2.apj	C:\Users\crunfola001\Desktop\NEW API\As sent to Nathalie
Arius_Sample3.apj	C:\Users\crunfola001\Desktop\NEW API\As sent to Nathalie
Arius_Sample4.apj	C:\Users\crunfola001\Desktop\NEW API\As sent to Nathalie

Run Sample | Recalculate

Tables to Retrieve		
Segment	Name	Type
PP AutoLiab	Ultimate Loss Based on Paid Loss Development	Method
Comm AutoLiab	Incurred Loss	Input
HO	Comparison of Ultimate Loss Estimates	Report
GL	Paid Loss Development	Exhibit

CONTROL | DATA

This example workbook processes two named range tables to demonstrate how to retrieve data from multiple Arius files. The example provides a simple Excel interface that allows you to easily change what files and what data are retrieved and placed on the DATA output tab, without any further programming.

There are two tabs in this example workbook, CONTROL and DATA.

The first table on the CONTROL tab, FILENAMES, identifies a list of Arius files from which to retrieve data. You may insert or remove rows to increase or decrease the number of Arius files (the key here is to increase/decrease the named range). Since the Arius install program is flexible and the user can choose any directory structure, we have leveraged an Excel API to determine the current location of the workbook so we can find the sample files for this example (we are assuming the sample files are in the same folder as the workbook). Replace the file name and path in order to specify your own files.

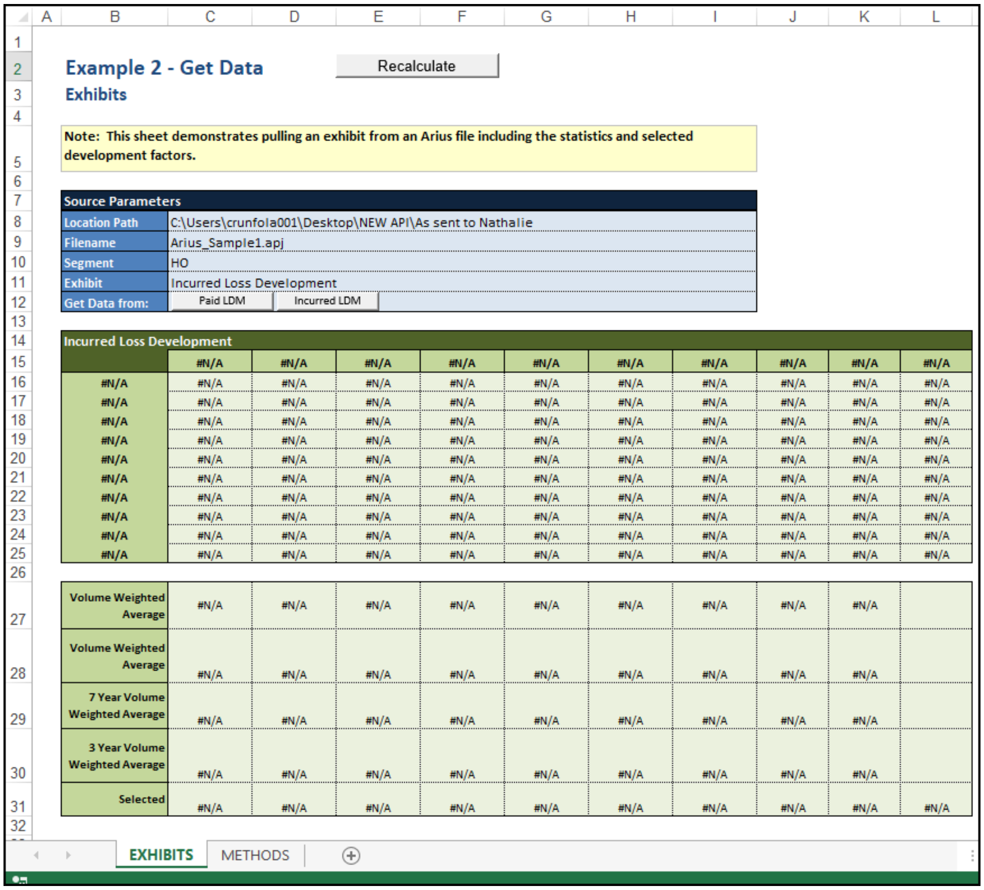
The second table on the CONTROL tab, TABLENAMES, identifies what data to retrieve from each Arius file specified in the FILENAMES list. Again, you can expand or shrink this table as appropriate.

Lastly, pressing the **Run Sample** button will execute the macro RunSample. When the macro executes, it will create two loops. The outer loop will open each file name specified in FILENAMES. The inner loop will retrieve each table name specified in TABLENAMES and write the results on the second tab of the workbook called DATA.

The result of this example will be a worksheet where the specified data is exported (and formatted using Excel functions).

Recalculate: When data within your Arius project file has changed, Excel will not perceive that this data has changed and will not automatically recalculate your Excel workbook. Press the **Recalculate** button to refresh the data in your Excel workbook from your Arius project file.

EXAMPLE 2: SPECIFIC GET DATA



NOTE:

Example 2 is the simplest of all the examples to learn how to wrap API calls, and to expose the VBA functions.

This example workbook demonstrates how to retrieve specific pieces of data from one Arius file. The example provides a simple Excel interface to specify different Arius files and data objects.

There are two tabs to review in this workbook. The first tab, EXHIBITS, demonstrates how to use the API to retrieve the various elements of one Arius development exhibit. Knowing how to pull one exhibit from an Arius file will allow you to design your own solution to retrieve multiple exhibits (including the different parts such as statistics and development factors). The second tab, METHODS, demonstrates how you might use the API to retrieve individual columns of data from multiple Arius methods (or Arius reports) and assemble them in one Excel table, say a Comparison of Ultimates report.

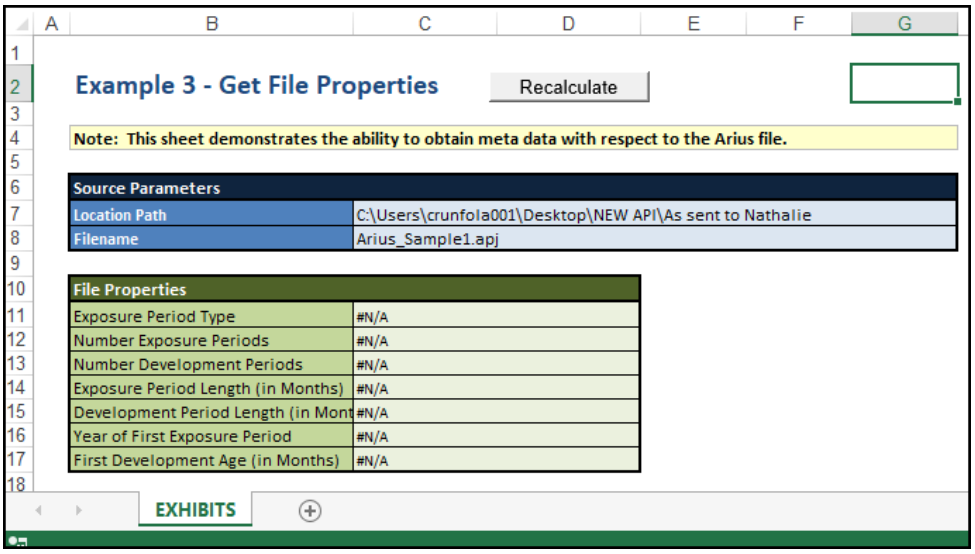
Each tab provides a named range, **Location**, where you can specify the Arius file path. A second named range, **FileName**, allows you to supply the file name from which to retrieve data. As mentioned in Example 1, we have leveraged an Excel API to determine the current location path of the workbook so we can find the sample file for this example.

Each of the tabs in this example workbook is populated with Arius-specific functions that will retrieve data from the specified Arius file. Two buttons exist on the EXHIBITS Tab: **Paid LDM** and **Incurred LDM**. Pressing these buttons will switch the formulas on the spreadsheet to change between Paid and Incurred data.

The METHODS tab has similar functionality and uses three buttons to switch between Segments (**HO**, **GL**, and **WC**). Pressing each button will adjust one of the input fields on the spreadsheet with a new parameter and all the functions will recalculate using this Segment.

Recalculate: When data within your Arius project file has changed, Excel will not perceive that this data has changed and will not automatically recalculate your Excel workbook. Press the **Recalculate b**utton to refresh the data in your Excel workbook from your Arius project file.

EXAMPLE 3: GET FILE PROPERTIES

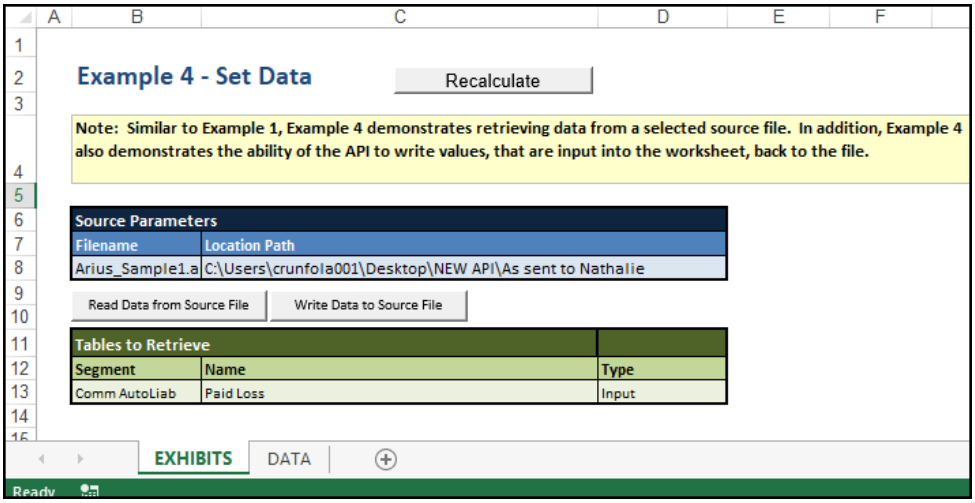


This simple example workbook contains one tab, EXHIBITS, and demonstrates how to retrieve general file property information from an Arius file (*.apj). The information is normally found under Project Settings in an Arius file, and includes the number and size of both exposure periods and development periods.

The worksheet allows you to specify a file path and then select one of the Arius files that exist in that directory (you will notice a drop down list available for the filename field). Once a file name has been selected, the embedded functions will execute and pull all the file information available from the specified file. Simply changing the file name will force Excel to recalculate all the functions.

Recalculate: When data within your Arius project file has changed, Excel will not perceive that this data has changed and will not automatically recalculate your Excel workbook. Press the **Recalculate** button to refresh the data in your Excel workbook from your Arius project file.

EXAMPLE 4: SET DATA



This example workbook contains two tabs, EXHIBITS and DATA. This workbook demonstrates how to use the API to both retrieve a triangle of data from Arius and also send data back to an input triangle in Arius. The example is sending only one piece of data back to Arius, but it can be modified to send several pieces of data back to Arius (or even populate most of an Arius file).

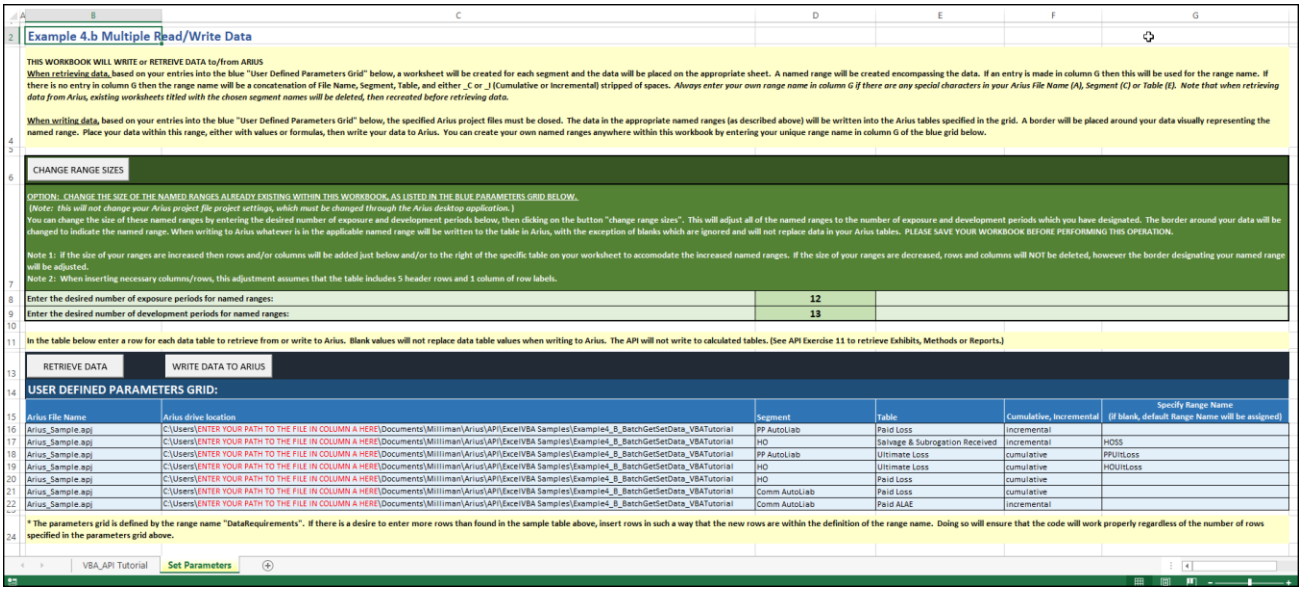
As in previous examples, on the EXHIBITS tab, you supply a file name and path, as well as the Segment and data object name. (The example is designed to work with Arius Input triangles but can be modified to handle any type of Arius object that accepts data.)

Pressing the **Read Data from Source File** button will insert a table on the DATA tab and the table will have a function that opens the specified file and retrieves the data. Also, notice that the results of this operation on the data tab contain an array formula.

The second button, **Write Data to Source File**, will insert the data from the DATA tab into the Arius file. Note that the **Read Data from Source File** button creates a triangle with an array formula, hence you will not be able to change any values in the table after you click on it. Therefore, we have provided a button on the data sheet that will copy the array formula on the data sheet and paste only the values into the named range, allowing you to edit the values (you could do this yourself, but we just made it easier). Pressing the **Read Data from Source file** again will quickly validate that the Set Data worked (or you can just open the Arius file using Arius and see for yourself).

Recalculate: When data within your Arius project file has changed, Excel will not perceive that this data has changed and will not automatically recalculate your Excel workbook. Press the **Recalculate** button to refresh the data in your Excel workbook from your Arius project file.

EXAMPLE 4B: GET AND SET DATA (INCLUDES VBA TUTORIAL)



Example 4.b Multiple Read/Write Data

THIS WORKBOOK WILL WRITE or RETREIVE DATA to/from ARIUS

When retrieving data, based on your entries into the blue "User Defined Parameters Grid" below, a worksheet will be created for each segment and the data will be placed on the appropriate sheet. A named range will be created encompassing the data. If an entry is made in column G then this will be used for the range name. If there is no entry in column G then the range name will be a concatenation of File Name, Segment, Table, and either _C or _I (Cumulative or Incremental) stripped of spaces. *Always enter your own range name in column G if there are any special characters in your Arius File Name (A), Segment (C) or Table (E). Note that when retrieving data from Arius, existing worksheets titled with the chosen segment names will be deleted, then recreated before retrieving data.*

When writing data, based on your entries into the blue "User Defined Parameters Grid" below, the specified Arius project files must be closed. The data in the appropriate named ranges (as described above) will be written into the Arius tables specified in the grid. A border will be placed around your data visually representing the named range. Place your data within this range, either with values or formulas, then write your data to Arius. You can create your own named ranges anywhere within this workbook by entering your unique range name in column G of the blue grid below.

CHANGE RANGE SIZES

OPTION: CHANGE THE SIZE OF THE NAMED RANGES ALREADY EXISTING WITHIN THIS WORKBOOK, AS LISTED IN THE BLUE PARAMETERS GRID BELOW.
(Note: this will not change your Arius project file project settings, which must be changed through the Arius desktop application.)
You can change the size of these named ranges by entering the desired number of exposure and development periods below, then clicking on the button "change range sizes". This will adjust all of the named ranges to the number of exposure and development periods which you have designated. The border around your data will be changed to indicate the named range. When writing to Arius whatever is in the applicable named range will be written to the table in Arius, with the exception of blanks which are ignored and will not replace data in your Arius tables. PLEASE SAVE YOUR WORKBOOK BEFORE PERFORMING THIS OPERATION.

Note 1: If the size of your ranges are increased then rows and/or columns will be added just below and/or to the right of the specific table on your worksheet to accomodate the increased named ranges. If the size of your ranges are decreased, rows and columns will NOT be deleted, however the border designating your named range will be adjusted.
Note 2: When inserting necessary columns/rows, this adjustment assumes that the table includes 5 header rows and 1 column of row labels.

Enter the desired number of exposure periods for named ranges:	12
Enter the desired number of development periods for named ranges:	13

In the table below enter a row for each data table to retrieve from or write to Arius. Blank values will not replace data table values when writing to Arius. The API will not write to calculated tables. (See API Exercise 11 to retrieve Exhibits, Methods or Reports.)

RETRIEVE DATA | WRITE DATA TO ARIUS

USER DEFINED PARAMETERS GRID:

Arius File Name	Arius drive location	Segment	Table	Cumulative, Incremental	Specify Range Name (if blank, default Range Name will be assigned)
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	PP AutoLiab	Paid Loss	incremental	
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	HO	Salvage & Subrogation Received	incremental	HOSS
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	PP AutoLiab	Ultimate Loss	cumulative	PPUltLoss
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	HO	Ultimate Loss	cumulative	HOUltLoss
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	HO	Paid Loss	cumulative	
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	Comm AutoLiab	Paid Loss	cumulative	
Arius_Sample.apj	C:\Users\ENTER YOUR PATH TO THE FILE IN COLUMN A HERE\Documents\Milliman\Arius\API\ExcelVBA Samples\Example4_B_BatchGetSetData_VBATutorial	Comm AutoLiab	Paid ALAE	incremental	

* The parameters grid is defined by the range name "DataRequirements". If there is a desire to enter more rows than found in the sample table above, insert rows in such a way that the new rows are within the definition of the range name. Doing so will ensure that the code will work properly regardless of the number of rows specified in the parameters grid above.

VBA_API Tutorial | Set Parameters

This example workbook was created to replicate the functionality of Example 4 with the alternative of multiple table reads/writes using VBA code that is easily understood by those who are new to programming or VBA. Yet, this workbook will function without any need to understand or access the VBA code. However, if desired, this workbook does provide a tutorial on how to use VBA with the Arius API for those who wish to learn. In addition, this workbook will allow you to change the sizes of named ranges around your data.

The workbook contains two tabs, VBA_API TUTORIAL and SET PARAMETERS. This workbook demonstrates how to use the API to retrieve one or many Data tables from Arius and send data back to a Data table in Arius. These tables could be columnar or triangular in shape.

Tip to Improve Performance:

This workbook allows for the retrieval and writing of one or multiple tables from one or multiple Arius project files. If this workbook is used to write a significant number of tables in a batch, you may experience performance issues. This code is written as a generic example where a different Arius project could be named on each row of the **Set Parameters** grid, so an Arius project is essentially opened and closed for each table requested. It is possible to dramatically speed up performance when retrieving a large batch of tables by limiting the list of requested tables to a single Arius project in a batch and modifying the VBA WriteDataModule to move the line of code shown below down 3 lines until it is just BELOW the code “Next i” rather than above. This will prevent excessive manipulation of the Arius project files.

PublicDeclarationsModule.AriusProject.WriteFile FullPath

NOTE:
Tip to improve performance when retrieving large batches of tables.

Using the Example 4B workbook to retrieve or write data from/to Arius

On the SET PARAMETERS tab, in the light blue **User Defined Parameters Grid**, supply file names and paths, as well as the Segments and data object table names you wish to retrieve from or write to Arius. Table and segment names must match Arius exactly. Long or abbreviated table names may be used. In

addition, you can specify cumulative or incremental data and a customized range name (especially helpful when the Arius table name contains a character that is not valid for an Excel range name).

The light blue **User Defined Parameters Grid** found on this tab has been assigned the named range DataRequirements through Excel. To list more or fewer tables for your batch, you can insert or delete rows from this grid and the code will function normally, as long as the range name DataRequirements continues to define the light blue area of grid rows.

Pressing the **Retrieve Data** button will create a new tab for each segment. Each table requested for a particular segment will be retrieved from Arius and will populate that segment's sheet. Unlike Exercise 4 after which this exercise is modeled, data *values* are displayed on the segment tab rather than array formulas.


NOTE:

If a tab for the same segment already exists in the workbook, the tab will be deleted from the workbook before data is retrieved.

A warning message is displayed allowing the user to cancel the retrieval process.

When retrieving data from Arius, range names will be assigned to the data section of the retrieved table (not including row or column labels) using the custom range names provided in column G of the **User Defined Parameters Grid** or, if column G is blank for a given row, then a unique default range name will be assigned for this data. Default range names will be a concatenation of Arius file name, segment, table, and either _C or _I. A border will be placed around the named range applied to your data.

When writing data to Arius, data existing within the named range specified for a row on your **User Defined Parameters Grid** (as described in the paragraph above) will be written to the table in Arius. This named range must match the size of your tables in your Arius project (same number of rows and columns).

Note: *Arius will not write to a calculated table.*

To write data to Arius, your Arius project must be closed and the data table you are writing in Excel must be the same size and shape as the data table in your Arius project. You will be notified if any of the Arius projects included in the light blue **User Defined Parameters Grid** are open, and the process will halt before any data is written to any of the Arius projects specified.

When writing to Arius, the range name for the retrieved data will be used to identify the data in the Excel workbook which is to be set in Arius when clicking **Write Data to Arius**. Retrieving data from Arius as a first step will set-up the current Arius structure (row/column labels) and range names for your use, if desired. Note that column and row headings are for reference only and are ignored when writing to Arius. However, you can create and populate your own named ranges on any sheet within the Excel workbook, specify the range name in column G and then write that data to Arius without first retrieving data from Arius.

Also, unlike Exercise 4, because the retrieved data is displayed as *values*, you can make changes to the data directly in the Excel named range for your data table and write this modified data to Arius. Although this workbook retrieves actual data values to the cells in Excel, if desired you can populate the Excel named ranges with formulas pulling data from elsewhere in your workbook to populatethe Excel triangles for writing to Arius.

Please be careful to create a copy of your Arius project (s) before using this workbook.

Modifying the size of named ranges

If you are using the same workbook to populate data tables in Arius each period, it will be necessary to adjust the size of the named ranges for the tables to accommodate new exposure and/or development periods. You can accomplish this in three ways: manually alter the named ranges, append a new

diagonal in Arius then retrieve the Arius data, or use the built-in feature of this workbook to change named range dimensions.

To use the built-in feature to change the size of specified Excel named ranges in the workbook, enter the number of exposure and development periods desired in cells D8 and D9 on the SET PARAMETERS worksheet then click the **Change Range Sizes** button.

This process will reference the light blue **User Defined Parameters Grid** to determine which named ranges to alter. If you have made an entry in column G of the **User Defined Parameters Grid** this named range will be altered. If column G is blank for a given row then the named range which is identified by a name concatenating the Arius file name, segment, table, and either _C or _I, will be altered.

The range dimensions will be changed to a rectangle with the number of rows in cell D8 and the number of columns in cell D9. Also, the border around the named range will be changed to encompass the new range. If the new range size is larger than the existing range size, then rows and columns will be inserted around the new range to avoid overwriting data in nearby cells. If the new range size is smaller than the existing range size, no cells will be removed from the workbook when resizing the named ranges. This process will resize named ranges on any sheet in the workbook where they might exist.

Note that this process will not resize the project settings within your Arius project.

Please be careful to create a copy of your Excel workbook before modifying your range sizes.

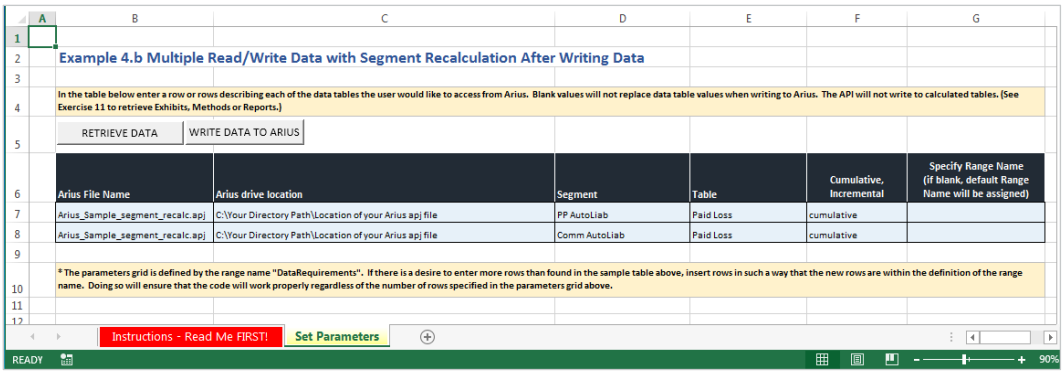
Updating only the last diagonal in Arius

To add only the last diagonal of a triangle to Arius, all cells of the Excel triangle except the last diagonal must be blank. Any blank values will be ignored when writing to Arius, preserving the values in the Arius triangle for the blank exposure/development periods. Please note that zero is not considered blank; a value of zero in the Excel triangle will write a zero to the Arius table.

Using this workbook as a VBA tutorial

The worksheet **VBA_API Tutorial** provides details on how to use Example 4B as a VBA and Arius API tutorial. Here you will find instructions about how to view VBA code in Excel and components that must be added to your Excel workbook to function with the Arius API. You will also be instructed how to open and review the heavily commented VBA code for retrieving data from Arius with step-by-step instructions about the code and logic employed. A lightly commented version of the same code is also provided so that you can follow the code and refer to the heavily commented version where you have questions. This is all explained in detail on the VBA_API Tutorial worksheet.

EXAMPLE 4B (VARIATION): GET AND SET DATA WITH RECALCSEGMENT



Example 4.b Multiple Read/Write Data with Segment Recalculation After Writing Data

In the table below enter a row or rows describing each of the data tables the user would like to access from Arius. Blank values will not replace data table values when writing to Arius. The API will not write to calculated tables. (See Exercise 11 to retrieve Exhibits, Methods or Reports.)

RETRIEVE DATA | WRITE DATA TO ARIUS

Arius File Name	Arius drive location	Segment	Table	Cumulative, Incremental	Specify Range Name (if blank, default Range Name will be assigned)
Arius_Sample_segment_recalc.apj	C:\Your Directory Path\Location of your Arius apj file	PP AutoLiab	Paid Loss	cumulative	
Arius_Sample_segment_recalc.apj	C:\Your Directory Path\Location of your Arius apj file	Comm AutoLiab	Paid Loss	cumulative	

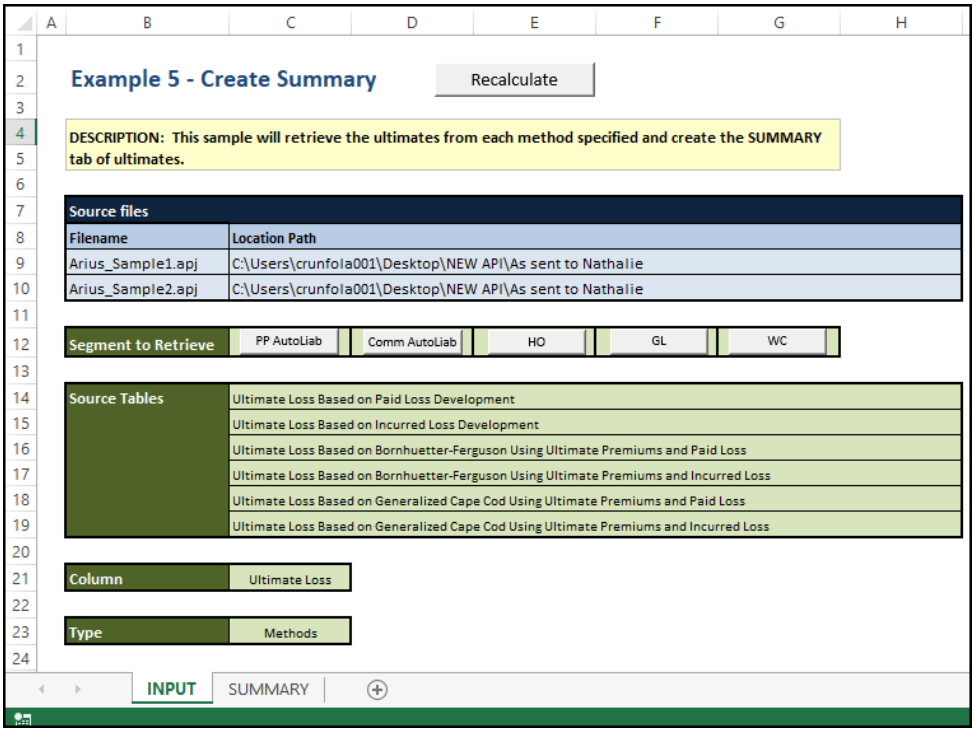
* The parameters grid is defined by the range name "DataRequirements". If there is a desire to enter more rows than found in the sample table above, insert rows in such a way that the new rows are within the definition of the range name. Doing so will ensure that the code will work properly regardless of the number of rows specified in the parameters grid above.

Instructions - Read Me FIRST! | Set Parameters

READY 90%

This variation of EXAMPLE 4B: GET AND SET DATA (INCLUDES VBA TUTORIAL) is identical except that it includes recalculation of all segments when each table is written, for those projects which take advantage of calculated segments in Arius. When data is updated in Arius, it is necessary to recalculate all of the segments to update these calculations. Within Arius, this is accomplished by clicking a button to recalculate all segments. Within the Arius API, this is accomplished by using the subroutine **RecalcSegments**. Using this example, segments are only updated as each table is updated in Arius. The segments are not recalculated when data is retrieved.

EXAMPLE 5: SUMMARY



This example workbook contains two tabs, INPUT and SUMMARY. This workbook demonstrates how to retrieve specific pieces of data from multiple Arius files and create a summary report in Excel. The interface allows you to change what files and what data are being retrieved. The example does not leverage any API or function calls on the spreadsheet, but instead demonstrates how to use the Arius API entirely using VBA functions.

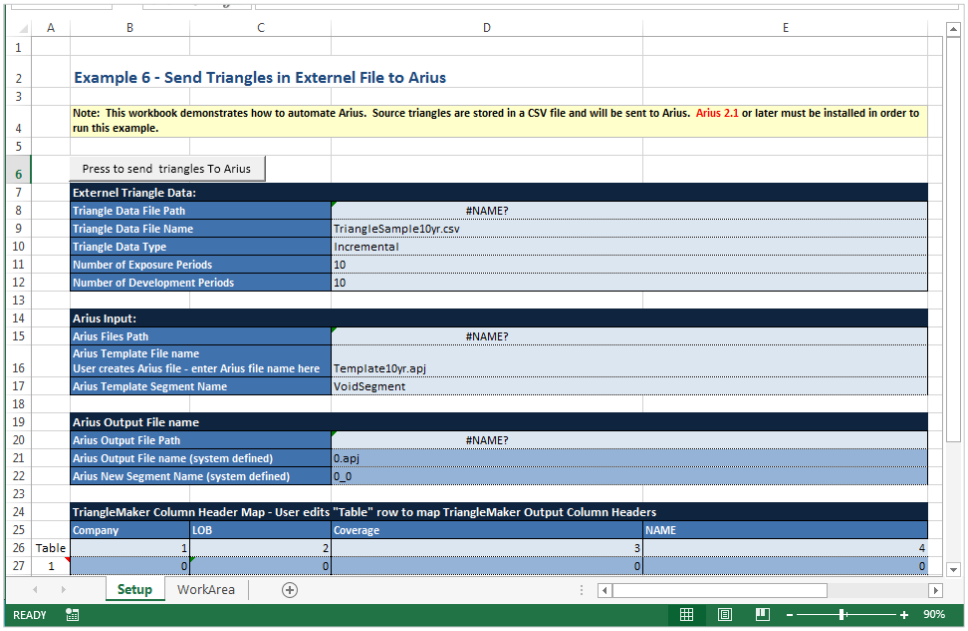
To make this example useful, we chose to demonstrate how to create a simple summary of ultimates from multiple Arius files. The example has many attributes similar to Example 1 (file list table, table list to process, segment names) but the output is very different. The example will demonstrate how to use the Arius API to extract one column of data from an Arius method table and then use Excel VBA to build a summary table containing all the ultimates processed.

Looking at the INPUT tab, you could identify multiple Arius files to be processed by specifying them in the Source Files table (again we are assuming the Arius files are in the same path as the workbook, so we are using a simple Excel API to identify the active directory). The Source Tables table allows you to identify what Arius method tables to use. The Column table identifies what common column to retrieve from each Arius Method (e.g., we are using the column titled Ultimate Loss since it is common to all Arius Methods and has values for this example). Lastly, we specify the Arius table type (e.g., Method) so the API knows which table to open in Arius (the example is designed to assume the data is a table of columns so you should limit your experiments to Arius Methods and Reports for this example). Pressing any of the **Segment to Retrieve** buttons will change the Segment parameter the GetData function passes to the API, invoking the function to retrieve the data from the specified Arius file and create a summary table on the SUMMARY tab/output sheet. (Note: results from previous runs

are currently cleared, but you can modify the code to keep appending the results below one another for each SegmentName).

Recalculate: When data within your Arius project file has changed, Excel will not perceive that this data has changed and will not automatically recalculate your Excel workbook. Press the **Recalculate** button to refresh the data in your Excel workbook from your Arius project file.

EXAMPLE 6: SEND TRIANGLES IN EXTERNAL FILE TO ARIUS



This workbook contains two tabs: SETUP and WORKAREA.

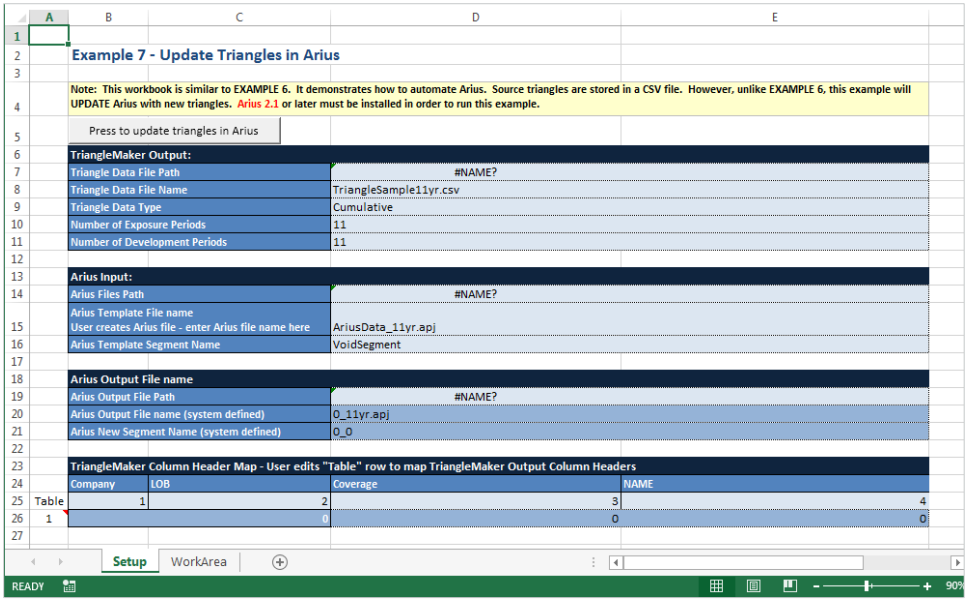
There are three VBA modules: DataModule, SetAriusData, and Main.

- The TriangleMaker Module is not needed for this example.
- This example is mostly the same as Arius_TriangleMaker_API_SendData.xlsm, but it skips the TriangleMaker step and works directly with a CSV file which contains the input triangles.

Sub-Folders/Files required to run:

- Subfolder Arius_Input – Arius Template file, Template10yr.apj, with a default segment "VoidSegment". Triangle data input file, TriangleSample10yr.csv.
- Subfolder Arius_Output –Arius output file path.

EXAMPLE 7: UPDATE TRIANGLES IN ARIUS



This workbook contains two tabs: SETUP and WORKAREA.

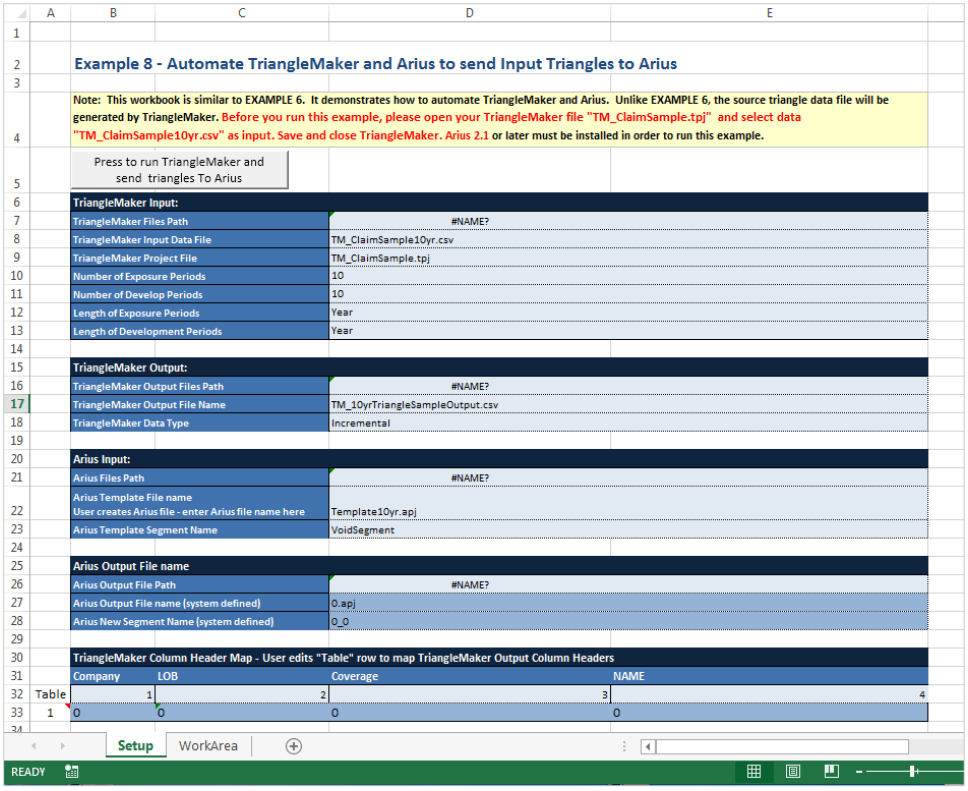
There are three VBA modules: DataModule; SetAriusData; and Main.

- The TriangleMaker Module is not needed for this example.
- This example is mostly the same as Arius_TriangleMaker_API_Update.xlsm, but it also skips the TriangleMaker step, working directly with the CSV file which contains the input triangles.

Sub-Folders/Files required to run:

- Subfolder Arius_Input – Arius Template file, AriusData_11yr.apj, with a default segment "VoidSegment". Triangle data input file, TriangleSample11yr.csv.
- Subfolder Arius_Output –Arius output file path.

EXAMPLE 8: AUTOMATE TRIANGLEMAKER TO SEND INPUT TRIANGLES TO ARIUS



Example 8 - Automate TriangleMaker and Arius to send Input Triangles to Arius

Note: This workbook is similar to EXAMPLE 6. It demonstrates how to automate TriangleMaker and Arius. Unlike EXAMPLE 6, the source triangle data file will be generated by TriangleMaker. Before you run this example, please open your TriangleMaker file "TM_ClaimSample.tpj" and select data "TM_ClaimSample10yr.csv" as input. Save and close TriangleMaker. Arius 2.1 or later must be installed in order to run this example.

Press to run TriangleMaker and send triangles To Arius

TriangleMaker Input:	
TriangleMaker Files Path	#NAME?
TriangleMaker Input Data File	TM_ClaimSample10yr.csv
TriangleMaker Project File	TM_ClaimSample.tpj
Number of Exposure Periods	10
Number of Develop Periods	10
Length of Exposure Periods	Year
Length of Development Periods	Year

TriangleMaker Output:	
TriangleMaker Output Files Path	#NAME?
TriangleMaker Output File Name	TM_10yrTriangleSampleOutput.csv
TriangleMaker Data Type	Incremental

Arius Input:	
Arius Files Path	#NAME?
Arius Template File name User creates Arius file - enter Arius file name here	Template10yr.apj
Arius Template Segment Name	VoidSegment

Arius Output File name	
Arius Output File Path	#NAME?
Arius Output File name (system defined)	0.apj
Arius New Segment Name (system defined)	0_0

TriangleMaker Column Header Map - User edits "Table" row to map TriangleMaker Output Column Headers

	Company	LOB	Coverage	NAME
Table	1	2	3	4
1	0	0	0	0

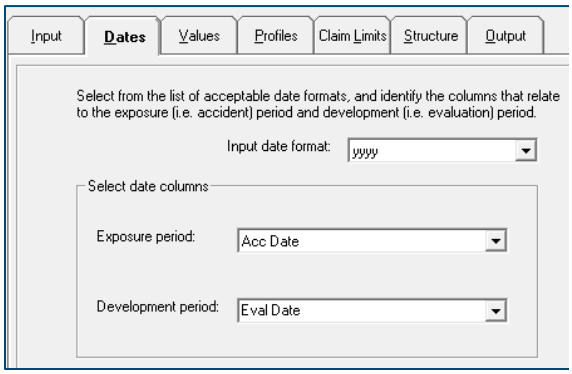
Setup | WorkArea

READY

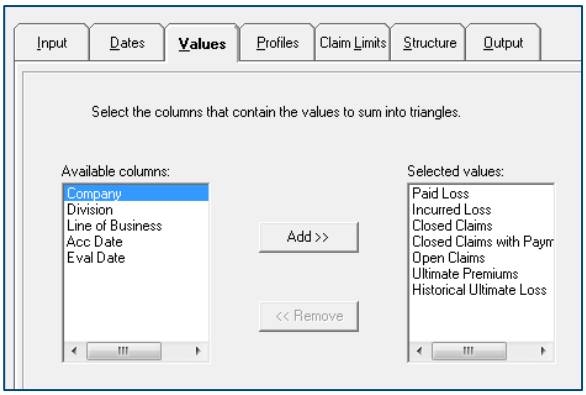
Before running this example:

- Open the file TM_ClaimSample.tpj (found in the …\ EXAMPLE8_TMSETINPUTDATA\ TM_Input folder) with TriangleMaker.
- Select the TM_ClaimSample10yr.csv file (in the same folder at the .TPJ file) as the CSV input file.
- Properly configure the various TriangleMaker tabs as shown below:

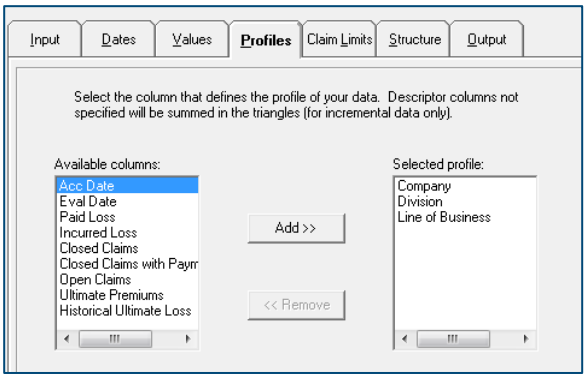
DATES TAB



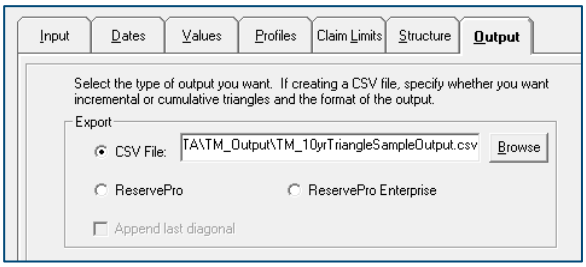
VALUES TAB



PROFILES TAB



OUTPUT TAB



- Choose the following path for the TM_10yrTriangleSampleOutput.csv output file: \Milliman\Arius\API\ExcelVBA Samples\EXAMPLE8_TMSETINPUTDATA\TM_Output.
- Save the TriangleMaker project file.

This workbook contains two tabs: SETUP and WORKAREA.

There are four VBA modules: TMAutomation, DataModule, SetAriusData and Main.

Sub-Folders/Files required to run:

- Subfolder Arius_Input

  Contains the Arius Template file, Template10yr.apj with a default segment VoidSegment.

- Subfolder Arius_Output

  Contains the Arius output file (once created).

- Subfolder TM_Input

  Contains the CSV Input File TM_ClaimSample10yr.csv and the TriangleMaker project file TM_ClaimSample.tpj.

- Subfolder TM_Output

  Contains the TriangleMaker output, no input file needed.

TMAutomation module

Subroutine GenerateTM

Purpose:

To automate TriangleMaker to process transactional data to generate triangles.

Details:

Creates TriangleMaker COM object and initializes it by providing the serial number.

```
Set sdk1 = New PWCTMCOM.ITMSDK
InitFlag = sdk1.Initialize("EB0F4B6D-011C-44AB-AFFB-B9408521735E")
```

Opens TriangleMaker Template File, TM_ClaimSample.tpj in this example.

```
result = sdk1.OpenProject(Range("TMLocation").value & "\" & Range("TMTemplate").value)
```

Updates TriangleMaker settings, see VBA for details.

Runs the TriangleMaker API to generate Triangles and save to output CSV file, output is saved to TM_Output folder in this example.

```
result = sdk1.Run()
```

Subroutine ClearTMData

Purpose:

Clears the contents and formats the WORKAREAtab.

Subroutine RetriveTMDataFiles

Purpose:

To retrieve the TriangleMaker generated output files from workbook.

Details:

Input Parameter is the file name with full path.

- Calls ClearTMData routine first to clear existing results, then Query data.

SetAriusData module

Subroutine WriteFile

Purpose:

To send individual triangles found in WORKAREA tab to Arius one at a time.

Details:

There is a formula in the worksheet that will determine the Arius file name and segment name.

- In this example, file name is Arius_Company Name.apj, segment name is Coverage, they change based on the working triangle (Table Index).
- When file name changes, it saves the Arius template file as a new Arius file.

```
AriusProject.SaveFileAs TemplateFile, AriusOutLocation & "\" & Filename
```

- When the Arius new segment name changes, it will copy the default segment DummySegment to a new segment called LOB_Coverage, and then save the file.

```
AriusProject.CopySegment AriusOutLocation & "\" & Filename, DummySegment
AriusProject.WriteFile AriusOutLocation & "\" & Filename
```

- Then it will send the triangles to Arius.

```
AriusProject.Data(FullPath & "\" & Filename, LOB, TableType, TableName, TabProperty) =
TriangleData.
```

  - TriangleData is the triangle to be sent to Arius.
  - The first argument FullPath & "\" & Filename is the working Arius file name with the full path.
  - The second argument, LOB is the working Segment Name.
  - TableType is the triangle data type.
  - TableName for the stochastic table names; check the StochasticTableAPIName list for details.

- Clears memory using the FlushFile function after the Arius Project file is saved.

```
AriusProject.FlushFile FullPath & "\" & Filename
```

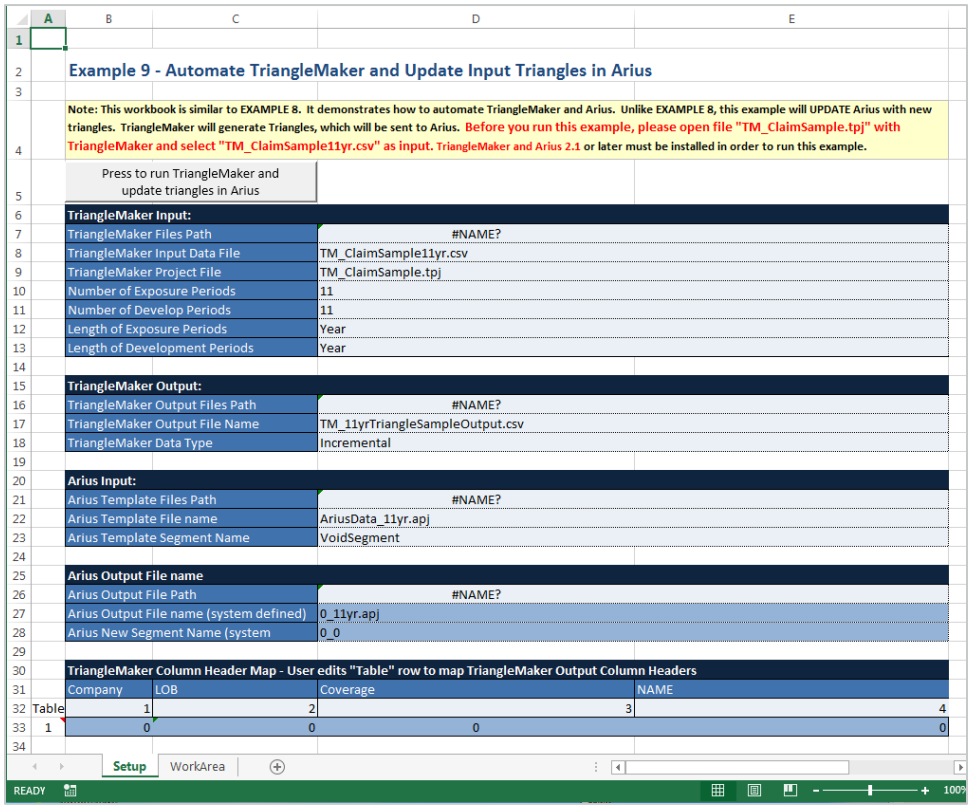
Main module

Purpose

User Defined Excel function apiPath, used to retrieve the working directory of the open workbook.

CreateAriusFiles – call all 3 modules to automate TriangleMaker, retrieve TriangleMaker output csv file from workbook and send triangles to Arius.

EXAMPLE 9: AUTOMATE TRIANGLEMAKER & UPDATE INPUT TRIANGLES IN ARIUS



Example 9 - Automate TriangleMaker and Update Input Triangles in Arius

Note: This workbook is similar to EXAMPLE 8. It demonstrates how to automate TriangleMaker and Arius. Unlike EXAMPLE 8, this example will UPDATE Arius with new triangles. TriangleMaker will generate Triangles, which will be sent to Arius. Before you run this example, please open file "TM_ClaimSample.tpj" with TriangleMaker and select "TM_ClaimSample11yr.csv" as input. TriangleMaker and Arius 2.1 or later must be installed in order to run this example.

Press to run TriangleMaker and update triangles in Arius

TriangleMaker Input:	
TriangleMaker Files Path	#NAME?
TriangleMaker Input Data File	TM_ClaimSample11yr.csv
TriangleMaker Project File	TM_ClaimSample.tpj
Number of Exposure Periods	11
Number of Develop Periods	11
Length of Exposure Periods	Year
Length of Development Periods	Year

TriangleMaker Output:	
TriangleMaker Output Files Path	#NAME?
TriangleMaker Output File Name	TM_11yrTriangleSampleOutput.csv
TriangleMaker Data Type	Incremental

Arius Input:	
Arius Template Files Path	#NAME?
Arius Template File name	AriusData_11yr.apj
Arius Template Segment Name	VoidSegment

Arius Output File name	
Arius Output File Path	#NAME?
Arius Output File name (system defined)	0_11yr.apj
Arius New Segment Name (system	0_0

TriangleMaker Column Header Map - User edits "Table" row to map TriangleMaker Output Column Headers

	Company	LOB	Coverage	NAME
Table	1	2	3	4
1	0	0	0	0

Setup | WorkArea

READY

This workbook contains two tabs: SETUP and WORKAREA.

- It is the similar to Arius_TriangleMaker_API_SendData.xlsm, the only difference is that the segments are included in Arius Template file and a different input file is used.

Sub-Folders/Files required to run:

- Subfolder TM_Input

  CSV Input File TM_ClaimSample11yr.csv, TriangleMaker Template file TM_ClaimSample.tpj.

- Subfolder TM_Output

  TriangleMaker output location, no input file needed.

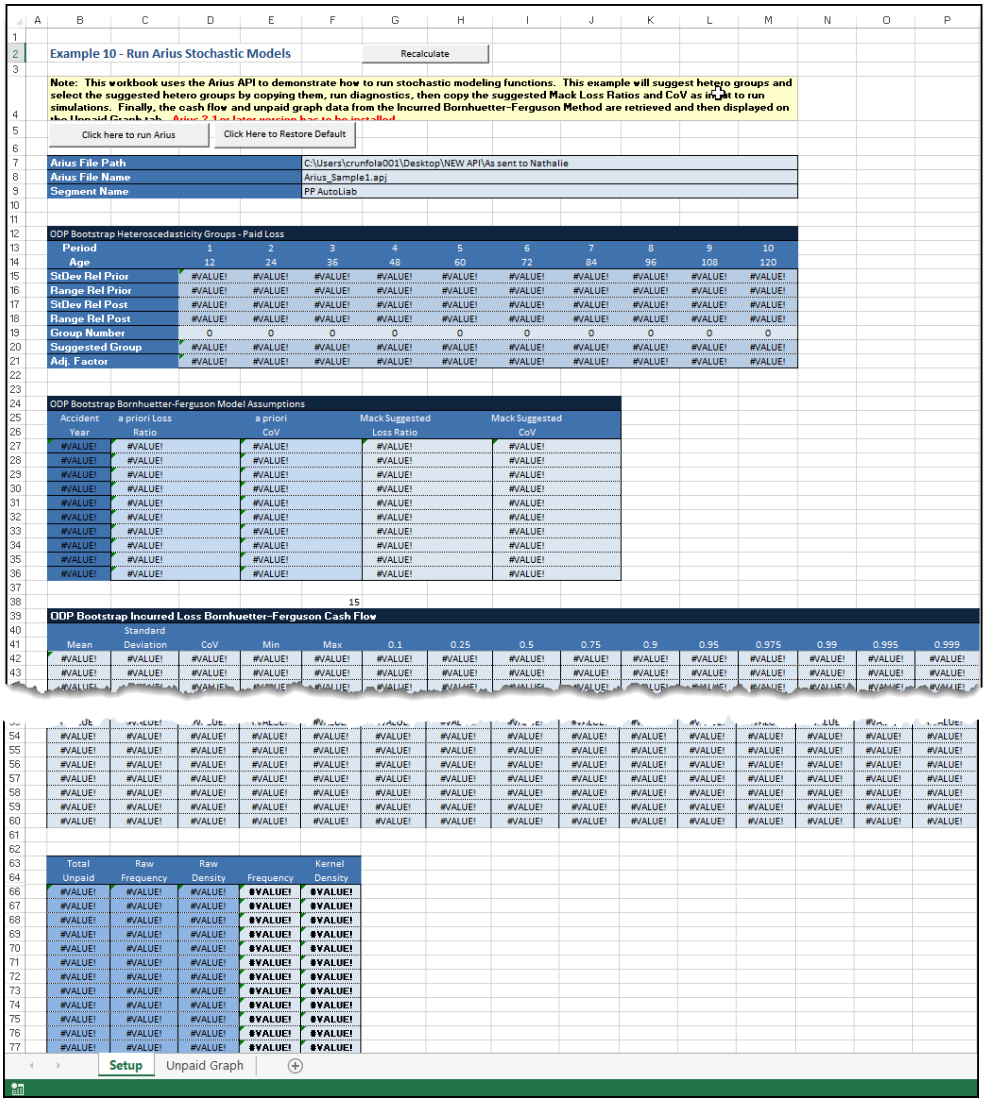
- Subfolder Arius_Input

  Arius Template file, AriusData_11yr.apj, that has all segments to work with in place.

- Subfolder Arius_Output

  Arius output file path.

EXAMPLE 10: RUN ARIUS STOCHASTIC MODELS



This workbook contains two tabs: SETUP and UNPAID GRAPH

This is an example of gettings data from Arius, setting data to Arius and Running Arius Stochastic modeling functions.

Sub-Folders/Files required to run:

- Arius_Sample1.apj

Function GetAriusTable

Purpose:

User defined VBA function to retrieve results from an Arius File

Subroutine RunArius

Purpose:

To send inputs/assumptions to an Arius File and run stochastic modelling functions.

Details:

- Save working Arius file Arius_Sample1.apj as Arius_Sample1_withRes.apj
- Run Arius to suggest hetro groups.

```
Call AriusProject.runservice(FolderName & "\" & NewFileName, Segment, "SuggestedHetero").
```

- Excel use embedded User define function to get the suggested hetero group number from Arius

```
GetAriusTable(FolderName,FileName,SegmentName,"ODP Paid","Hetero","SuggGrp").
```

- Select suggested hetero group.

```
Range("HeteroGrp").Value = Range("SuggHeteroGrp").Value
AriusProject.Data(FolderName & "\" & FileName, Segment, "ODP Paid", "Hetero", "Grp") =
Range("HeteroGrp").Value
```

- Run Diagnostics

```
Call AriusProject.runservice(FolderName & "\" & NewFileName, Segment, "Diagnostics").
```

- Copy Mack suggested Loss Ratio and CoV.

```
AriusProject.Data(FolderName & "\" & NewFileName, Segment, "ODP Paid",
"BF_ModelAssumptions", "PriorLR") = WorksheetFunction.Transpose(Range("PriorLR").Value)  '
Copy Prior LR from suggested value.

AriusProject.Data(FolderName & "\" & NewFileName, Segment, "ODP Paid",
"BF_ModelAssumptions", "PriorCoV") = WorksheetFunction.Transpose(Range("PriorCoV").Value) '
Copy Prior CoV from suggested value.
```

- Run Simulation

```
Call AriusProject.runservice(FolderName & "\" & NewFileName, Segment, "Simulation")
```

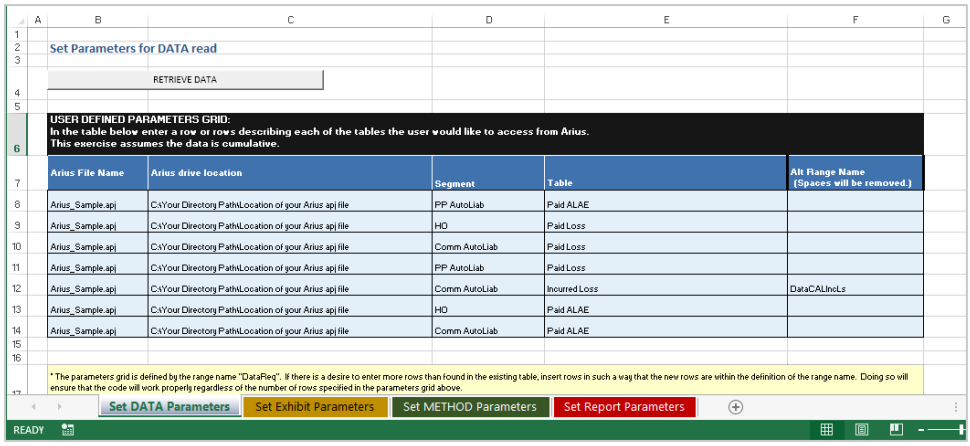
- Save Arius file:

```
AriusProject.WriteFile FolderName & "\" & FileName
```

Ultimately, the Excel embedded functions retrieve results from the Arius file, including: Cash Flow and Unpaid Graph data from the Incurred Bornhuetter-Ferguson Method.

Recalculate: When data within your Arius project file has changed, Excel will not perceive that this data has changed and will not automatically recalculate your Excel workbook. Press the **Recalculate** button to refresh the data in your Excel workbook from your Arius project file.

EXAMPLE 11: BATCH GET DATA BY TYPE: DATA, EXHIBIT, METHOD, AND REPORT



This example workbook is set up to retrieve (Get) multiple data objects for all data types except scalars and will function without any need to understand or access the VBA code.

This workbook contains four tabs: SET DATA PARAMETERS, SET EXHIBIT PARAMETERS, SET METHOD PARAMETERS, and SET REPORT PARAMETERS.

- This example demonstrates how to use the API to retrieve data input triangles/columns, methods, reports, individual columns from methods or reports, LDFs from Exhibits, or specified rows of statistics from Exhibits. These objects could be triangles, rows, or columns.

On the SET PARAMETERS tab for each object type, supply file names and paths, as well as the Segments and table names that you wish to retrieve from Arius. Table names must match Arius exactly, using either the long or abbreviated name. Range names will be assigned to the retrieved object's data using a unique default range name, or you can specify a customized range name. Each parameters grid has been assigned a named range through Excel. To list more or fewer tables in a grid, insert or delete rows in the grid and the code will function normally as long as the range name continues to define the list area of the grids' rows.

Data type objects:

Data is assumed to be cumulative. See Example 4B for an example of how to specify cumulative or incremental.

Exhibit type objects:

Exhibit Statistic may be specified. If specified, the retrieval will return only the requested row of statistics from the exhibit. If not specified, the triangle of LDFs for the exhibit will be returned.

Method and Report type objects:

An individual column may be specified. If specified, the retrieval will return only the requested column. If not specified, the entire method or report will be returned.

Pressing the **Retrieve** button will create a new tab for each segment, with the tab name identifying the data type (Data, Exhibit, Method, or Report). Each object requested for this segment and type will be retrieved from Arius and will populate the appropriate worksheet. If the range name (either user specified or default) already exists in the workbook, then this range will be cleared before data is retrieved from Arius. A warning message is displayed allowing the user to cancel the retrieval process.

Object values, rather than array formulas, are displayed on the worksheets. When worksheets are deleted, the range names remain so that external references to these range names will not be broken.

Tip to Improve performance:

This workbook allows for the retrieval of one or multiple tables from one or multiple Arius projects. If this workbook is used to get a significant number of tables in a batch you may experience performance issues. This is because the code is written as a generic example where a different Arius project could be named on each row of the **Set Parameters** grid. Therefore, an Arius project is essentially opened and closed for each table requested. It is possible to dramatically speed up performance when retrieving a large batch of tables by limiting the list of requested tables to a single Arius project in a batch and modifying the VBA for each object type as shown below. This will prevent excessive manipulation of the Arius project files and enhance performance.

NOTE:

Tip to improve performance when retrieving large batches of tables

Code #1 should be identified in the DataModule, MethodModule, ExhibitModule, and ReportModule. Replace Code #1 with Code #2.

Code #1

```
For i = 2 To ParamRows
Segment = Params(i, 3)
Worksheets(Segment & " DATA").Select
Range("A1").Value = ""

Filename = Params(i, 1)
Location = Params(i, 2)
FullPath = Location & "\" & Filename
AriusProject.FlushFile (FullPath)

Next i
```

Code #2

```
For i = 2 To ParamRows
Segment = Params(i, 3)
Worksheets(Segment & " DATA").Select
Range("A1").Value = ""
Next i
Filename = Params(1, 1)
Location = Params(1, 2)
FullPath = Location & "\" & Filename
AriusProject.FlushFile (FullPath)
```

4. The root of all API calls: the AriusSystem object

CREATE AN INSTANCE OF THE ARIUS SYSTEM OBJECT (ARIUSSYSTEM)

Before you can begin using any of the exposed Arius APIs, you must first instantiate at least one instance of Arius by creating a variable of type AriusSystem. Instantiating an AriusSystem object by creating an AriusSystem variable starts a hidden copy of Arius (note that it is not visible on screen, and some of Arius' full functionality is not available).

For those just getting started, we recommend that you do this at the very top of your module so the variable has global scope and will be available to all functions in the entire module. You can see this in each of our examples by looking at the top line in the VBA module.

```
Dim AriusProject as New AriusSystem
```

Alternatively, as you develop more advanced modules, you are encouraged to create local scope variables of type AriusSystem that go in and out of scope with either the module or routine. One way to do this is to simply dimension a variable of AriusSystem inside a subroutine. (There is no reason to create multiple instances of AriusSystem, as AriusSystem allows you to work with multiple data files.)

GLOBAL SCOPE INSTANCES VS. LOCAL SCOPE INSTANCES OF ARIUSSYSTEM

The main reason to create local scope instances of AriusSystem is for automated file management and garbage collection. Almost every property and method exposed by AriusSystem requires a file name to be identified. If that file is not currently loaded into memory, AriusSystem will open the file, load it into memory and close the file (similar to using File Open in Arius). This file data will remain in memory until you either call FlushFile or the AriusSystem object goes out of scope.

A routine that processes a large number of files using a global instance of AriusSystem will keep each instance of the file loaded in memory for the life of the variable AriusSystem if you do not call CloseFile. However, if you use a local instance of AriusSystem to process each file, AriusSystem will be destroyed when it goes out of scope and it will automatically purge the Arius file from memory, resulting in more efficient memory management.

5. Opening and closing files with FlushFile

Almost every method and property exposed from the Arius API requires a file name to be supplied. The internal code will check if the file is already open and if not, the file will be opened, loaded into memory, and then closed. When processing a group of files, remember that each file will remain in memory.

We provide several ways to clear these files from memory when you finish with them.

- The first and easiest way is to call **FlushFile**. FlushFile will delete the file from memory. You can include a call to FlushFile in your logic at the appropriate time for your function.
- The second way to remove files from memory is to let the AriusSystem object go out of scope, as previously discussed. Using GlobalScope (where the AriusSystem object is available for the life of the workbook) would necessitate calls to FlushFile. Using local scope (where the AriusSystem is instantiated and deleted within a function) would limit the lifetime of the files in memory; however, local scope instances of AriusSystem might still require calls to FlushFile if the routine is processing a large number of files.

6. Considerations when naming segments

Arius incorporates segment names into the names of PDF files it creates, to make sure your segment data is uniquely identified. In addition, segment names are used for Excel worksheet names. They can also be incorporated into Excel defined range names in the API sample Excel workbooks and perhaps in your own Excel workbooks.

When Arius creates Excel and Acrobat files, Windows' standard naming conventions must apply. Therefore, it is important to consider these Microsoft constraints when choosing your segment names.

- Excel worksheet names (i.e., tab names) have a maximum length of 31 characters. Arius' Export to Excel and the API examples provided with Arius use the segment names as Excel worksheet names. Be aware of this Excel constraint when naming your segments, as characters beyond the 31st will be truncated in any resulting XLSX tab names.
- The following special characters and words violate Windows' naming conventions for file names, Excel worksheet names, and Excel defined range names. They should not be used in segment names:

< (less than)	\ (backslash)
> (greater than)	| (vertical bar)
: (colon)	? (question mark)
" (double quote)	* (asterisk)
/ (forward slash)	Also, avoid using a period.

NOTE:

Segment names are case sensitive when using the API.

- The following Windows reserved words should be avoided as separate words in your segment names: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9.
- If you intend to use your segment name in an Excel defined range name, or if you will be using the Excel API examples provided with Arius, then the following rules are recommended when choosing your segment name:
  - Begin your segment name with a text character, not a number or special character.
  - Only four special characters are allowed in Excel's defined range names: underline, question mark, period, and back slash. However, all except the underline character violate file naming conventions, therefore only the underline character is recommended as a special character for use within your segment names.
  - Do not begin your segment name with something that could be interpreted as an Excel cell address. For example, R2_Mysegment will return an Excel error as a defined range name because R2 is an Excel cell address.
  - Note: API examples provided with Arius use the segment name to create Excel defined names. However, EXAMPLE4_B_BATCHGETSETDATA_VBATUTORIAL.xlsm and EXAMPLE11_BATCHGET_DataMethodReportExhibit.xlsm allow you to choose unique range names which do not incorporate the segment name. Also, in the remaining API examples and your own VBA code it is possible to strip all special characters from a segment name prior to incorporating into a defined range name.

7. Properties vs. methods

AriusSystem exposes all of its functionality as either a property or a method. In general, properties represent information whereas methods represent actions. A property can be regarded as a variable exposed by Arius and a method can be regarded as a function exposed by Arius. Generally speaking, methods require you to pass in one or more variables as parameters of the function call to get and set data.

Properties and methods require an AriusSystem object. Here are some examples of properties and methods:

```
Dim AriusProject as AriusSystem                          ’starts hidden Arius instance

Dim AsOfDate as String

AriusProject.OpenFile(“C:\Sample1.apj”)                  ‘this is a method of AriusSystem

Eval_Date= AriusProject.ValDate (“C:\Sample1.apj”)       ‘example retrieving a property value

AriusProject.Data(…) = MyRange.value                     ‘example changing value of a
                                                          property

AriusProject.CloseFile(“C:\Sample1.apj”)                  ‘this is a method of AriusSystem

AriusProject.FlushFile(“C:\Sample1.apj”)                  ‘this is a method of AriusSystem
```

8. Checking for errors

Each of the API methods and properties are capable of throwing an error if one should occur. In order to test for errors, you need to check the Err.Number property. If the API call is successful, Err.Number will equal ZERO, otherwise Err.Number will contain the error code.

Since error codes do not tell us a lot (other than an error occurred) we also provide an API call to return the error message associated with the error code.

```
Property ErrorMessage(long ErrorNumber) As String
```

You may use this API to retrieve the error message string for any error code. Below is an example:

```
Dim AriusProject as New AriusSystem
On Error Resume Next
'Make a call to the API
If Err.Number <> 0 Then

    MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

End If
```

Alternatively, you may decide to use On Error Goto if you decide you do not need to check for an error on every API call. It is completely up to you to decide how robust you want to make your code. Here is a list of the current error types (ErrorType):

E_UNKNOWN	E_FILE_NOT_RECOGNIZED
E_FILE_OPEN_FAILURE	E_DATA_SOURCE_NOT_RECOGNIZED
E_DICTIONARY_LOAD_FAILURE	E_TABLE_TYPE_NOT_RECOGNIZED
E_NAME_NOT_RECOGNIZED	E_TABLE_PART_NOT_RECOGNIZED
E_TABLE_NOT_FOUND	E_WRONG_FILE_TYPE

These are defined as constants in the IDL file and are recognized by VBA when you add a reference to the AriusAPI 2.0 Type Library. These are not the actual error codes returned by Err.Number in VBA. To get that exact value, the user needs to use the ErrorCode property and pass one of these constants as a parameter. Here is sample code that uses both ErrorMessage and ErrorCode to show all the possible error messages.

```
Dim AriusProject As New AriusSystem
Sub ShowAllErrorMessages()

    Dim etype(0 To 10) As ARIUSAPILib.ErrorType
    etype(0) = E_DATA_SOURCE_NOT_RECOGNIZED
    etype(1) = E_DICTIONARY_LOAD_FAILURE
    etype(3) = E_FILE_NOT_RECOGNIZED
    etype(4) = E_FILE_OPEN_FAILURE
    etype(5) = E_NAME_NOT_RECOGNIZED
    etype(6) = E_TABLE_NOT_FOUND
    etype(7) = E_TABLE_PART_NOT_RECOGNIZED
    etype(8) = E_TABLE_TYPE_NOT_RECOGNIZED
    etype(9) = E_UNKNOWN
    etype(10) = E_WRONG_FILE_TYPE
    For i = 0 To 10

        MsgBox AriusProject.ErrorMessage(AriusProject.ErrorCode(etype(i)))

    Next i

End Sub
```

9. Summary list of API methods and properties

Almost every API Method and property requires a filename (fully qualified with drive letter and path) as the first parameter. This is needed because the API allows multiple files to be in memory at the same time. The API was designed to simplify things for the developer. If the filename specified does not exist in memory, the filename will be automatically opened and loaded into memory (internally the API will call OpenFile and CloseFile automatically as necessary). You will be responsible for calling WriteFile and FlushFile when you wish to save the changes to the file and remove it from memory when done.

Once the file is loaded into memory, all operations will occur on the file in memory. Any changes to the files in memory will need a call to Write Files to save the changes.

Note: This list includes all functional components exposed via the API. Methods and properties visible within the VBA editor but not listed here are remnants from legacy systems and are not currently functional or supported.

METHODS

METHOD NAME	PARAMETERS	DESCRIPTION
CloseFile	(filename As String)	Closes the specified file. Only needed for files opened with OpenFile. File will still remain in memory and will require call to FlushFile to purge file from memory.
CopySegment	(filename As String, SegmentName As String, NewSegmentName As String)	Makes a copy of the specified segment in the specified file.
DeleteSegment	(filename As String, SegmentName As String)	Deletes the specified segment from the specified file.
FlushFile	(filename As String)	Purges the specified file from memory. Does not delete it from the disk.
OpenFile	(filename As String)	Opens the specified file and reads contents into memory. Be sure to call CloseFile when done.
RunService	(filename As String, [SegmentName As String], ServiceName As String)	Run the specified service for the specified segment in the specified file. If SegmentName is null, then the service will be run for all segments found in the file.
SaveFileAs	(filename1 As String, filename2 As String)	Makes a copy of the source file. The source file does not need to be previously opened or loaded into memory. Calling SaveFile will rename the file in memory so only one file will exist (as opposed to two). You will need to call FlushFile on the renamed file.
WriteFile	(filename As String)	Writes the contents of the file in memory to disk. Overwrites what was there previously. WriteFile will automatically call OpenFile and CloseFile if necessary (assuming the file is already loaded in memory).

PROPERTIES

PROPERTY NAME	PARAMETERS	RETURN TYPE	DESCRIPTION
ColumnLabels	(filename As String, SegmentName As String, TableType As String, TableRef, [PartName])	Array	Read only. Returns the text representing the column labels for the table or table subpart specified.
Data	(filename As String, SegmentName As String, TableType As String, TableRef, [PartRef])	<depends on request>	Sets and gets specified data for specified object.
DevelopmentPeriodLength	(filename As String)	Long	Read only. Gets the development period length from the specified file.
ErrorCode	(type As ErrorType)	Long	Read only. Returns the exact error code for a particular error type. This is useful when you want to check for a particular kind of error.
ErrorMessage	(ErrNum As Long)	String	Read only. Returns a formatted error message based on the error code supplied as a parameter.
ExposureDate	(filename As String)	Date	Read only. Retrieves the date of the first exposure period from the Arius file in PROJECT SETTINGS | DATA STRUTURE | DATE PARAMETERS.
ExposurePeriodLength	(filename As String)	Long	Read only. Retrieves the length of the exposure periods in an Arius file.
FirstDevelopmentMonth	(filename As String)	Long	Read only. Returns the month number of the first development period.
FirstExposureYear	(filename As String)	Long	Read only. Returns the year of the first exposure period.
HasPrior	(filename As String)	Long	Read only. Returns zero if no prior row, otherwise a nonzero value indicates there is a prior row.

PROPERTY NAME	PARAMETERS	RETURN TYPE	DESCRIPTION
LenLastCalPeriod	(filename as String)	Long	Read Only. Returns the length (in months) of the last calendar period.
NumColumns	(filename As String, SegmentName As String, TableType As String, TableRef, [PartRef])	Integer	Read only. Returns the number of columns for any table in a particular file. Will also return the number of columns for the subpart specified for a particular table (e.g., Exhibit statistics).
NumDevelopments	(filename As String)	Long	Read only. Returns the number of development periods specified for a particular Arius file.
NumExposures	(filename As String)	Long	Read only. Returns the number of exposure periods specified in an Arius file.
NumRows	(filename As String, SegmentName As String, TableType As String, TableRef, [PartName])	Integer	Read only. Returns the number of rows for the specified table in an Arius file. When PartName is supplied, will return the number of rows for that subpart of the specified table (e.g., Statistics).
PeriodType	(filename As String)	String	Read only. Returns whatever information is stored in PROJECT SETTINGS | DATA STRUTURE | EXPOSURE PERIOD TYPE
RowLabels	(filename As String, SegmentName As String, TableType As String, TableRef, [PartName])	Array of Strings	Read only. Returns a string array with the row labels for a particular table in an Arius file. When PartName is supplied, it will return the row labels for that subpart of the table (e.g., Exhibit Statistics).
Scaling Factor	(filename As String)	String	Read only. Returns the scaling factor assigned inside an Arius file.
Segments	(filename As String)	Array of Strings	Read only. Returns a string array with the segments in an Arius file.
TableIndexes	(filename As String, TableType As String)	Array of Integers	Read only. Returns an array of integers containing all the indexes for a particular table type in the file specified. Any of the values returned can

PROPERTY NAME	PARAMETERS	RETURN TYPE	DESCRIPTION
			then be used for the TableRef parameter.
TableNames	(filename As String, TableType As String)	Array of Strings	Read only. Returns an array of strings representing all the names of all tables for a particular table type.
TableParts	(filename As String, TableType As String, TableName As String)	Array of Strings	Read only. Returns an array of strings representing all of the table parts of the specified table type for the specified table name.
TableInfo	(filename As String, TableType As String)	Array of Integers and Strings	Read only. Returns a two-dimensional array containing information on the table names and indexes in the particular file.
TableTypes	(filename As String)	Array of Strings	Read only. Returns an array of strings representing each of the table types.
ValDate	(filename As String)	String	Read only. Returns a string representing the evaluation date of the Arius Project File.

10. API details

PARAMETERS OF ARIUS API METHODS & PROPERTIES

All properties and methods supported by the API accept parameters. Since the API is designed to provide access to Arius files, it is required that every function accept the filename as the first parameter. The API string parameters are not case sensitive. The following is a list of common parameters that are shared by multiple properties and methods.

Filename as string

Most properties and methods require the filename as the first parameter. It must contain the fully qualified path name of the file. The acceptable extension when selecting the file must be the Arius extension (*.apj).

C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.apj

TableType as string

The following are valid values for this parameter:

“Input”
“Exhibit”
“Method”
“Report”
“Scalar”

TableRef as variant

This parameter requires either a number or a string type to be passed into the receiving function. In the case of a number, it is assumed that the number represents the index number of the table. If a string is used, then this parameter would contain the name of the desired table. The string can be either the Arius table full name or the abbreviated name.

[PartRef] as variant

This is an optional parameter that can be used to expand on or drill down into a particular set of table values. For example, you can use this parameter to access the selections in an Arius exhibit or an individual column in a method or report. In the case of triangular data, this parameter can also be used to specify whether to return the data incrementally or cumulatively. The following are valid entries for this parameter:

Inputs

"Incremental" (return data incrementally; this is the default for triangle data)

"Cumulative" (return data cumulatively)

"IgnoreBlanks" (ignore blank cells in source data; set only cells with valid amounts;
see Data section of this document for details)

"Cumulative,IgnoreBlanks"

"Incremental,IgnoreBlanks"

Exhibits

"Statistics"

"TFA" (tail factor analysis)

"Selected" (selected development factors)

"Selected Interpolated" (selected interpolated development factors)

Individual exhibit statistics

This is not a complete list, as the number of combinations is endless; treat these as examples.

"Cumulative Selected"

"Ratio to Ultimate"

"Selected Interpolated"

"Cumulative Selected Interpolated"

"Ratio to Ultimate Interpolated"

"Avg"

"Average Excluding High/Low"

"Volume Weighted Average"

"3 Year Average"

Selected Interpolated development factors:

Arius can fit curves to your Selected Development Factors to help you estimate Selected Interpolated Development Factors. You can use GetData with the API to retrieve the Selected Interpolated Factors, as well as the related Cumulative Interpolated and Ratio to Ultimate Interpolated Factors.

However, for certain technical reasons, the API cannot retrieve the fitted curves themselves. Specifically, you cannot use GetData to retrieve the following interpolation statistics from an exhibit:

Linear – Cumulative

Linear – Ratio to Ultimate

Exponential – Cumulative

Exponential – Ratio to Ultimate

Weibull – Cumulative

Weibull – Ratio to Ultimate

Inverse Power – Cumulative

Inverse Power – Ratio to Ultimate

EXAMPLE CODE USING SUBROUTINES AND FUNCTIONS

You will notice that many of the sample code fragments will be implemented using a Function while others are implemented via a subroutine. This is a matter of style and how you might need to use it.

- Functions were chosen because they can be embedded within a spreadsheet.
- Subroutines cannot be embedded as part of formula on a spreadsheet and must be called by another subroutine.

CloseFile

Syntax:

Sub CloseFile(filename As String)

Description:

Closes the specified file. Only needed for files opened with OpenFile. File will still remain cached in memory and will require call to FlushFile to purge file from memory.

Return:

Nothing is returned.

Applies to:

*.apj files

VBA example:

```
Sub CloseFile (Filename As String)

On Error GoTo errHandler

    AriusProject.CloseFile Filename
    Exit Sub


errHandler:

    MsgBox "There was an error closing the file " & Filename
    Resume Next

End Sub
```

ColumnLabels

Syntax:

Property ColumnLabels(filename As String, SegmentName As String, TableType As String, TableRef, [PartName]) As Variant

Description:

Read-only. Returns the text representing the column labels for the table or table subpart specified.

Return:

Variant containing an Array of Strings

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem

Function GetColumnLabels(Filename As String, SegmentName As String, TableType As String, TableName As String)
    On Error Resume Next
    GetColumnLabels = AriusProject.ColumnLabels(Filename, SegmentName, TableType, TableName)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Entering the following as an Excel array formula:

```
= GetColumnLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL", "Input", "Paid Loss")
```

returns a string array with the column labels of the Segment in filename.

NOTE:

Since all the factors use the same labels, we can query any factor for the labels; therefore, we hard-coded the routine to use the GL segment and the Paid Loss input table

CopySegment

Syntax:

Sub CopySegment(filename As String, SegmentName As String, NewSegmentName As String)

Description:

Makes a copy of the specified segment in the specified file and creates a new segment using NewSegmentName.

CopySegment should always be immediately followed with WriteFile and FlushFile to avoid unexpected results.

For example:

```
AriusProject.CopySegment FileName, SegmentName, NewSegmentName
AriusProject.WriteFile FileName
AriusProject.FlushFile FileName
```

NOTE:

Alway follow CopySegment with WriteFiles and FlushFile

Return:

Adds a new segment.

Applies to:

*.apj files

VBA example:

```
Sub CopySegment(Filename As String, SegmentName As String, NewSegmentName As String)

On Error GoTo errHandler

    AriusProject.CopySegment(Filename, SegmentName, NewSegmentName)
    AriusProject.WriteFile(FileName)
    AriusProject.FlushFile(FileName)
    Exit Sub


errHandler:

    MsgBox "There was an error copying the segment in the file " & Filename
    Resume Next

End Sub
```

Data

Syntax:

Property Data (filename As String, SegmentName As String, TableType As String, TableRef, [PartRef]) As Variant

Description:

Gets and sets the specified data for an Arius object. By specifying the optional parameter (PartRef), the user can drill down into exhibits, methods, and reports. This property when being read always returns a two-dimensional array containing the data values from the Arius object referred to by the parameters of the Data property. When using the write version of the property, the value being written must also be a two dimensional array containing either numbers or empty strings. By default, empty strings in the source XLS file will be translated to blanks in the Arius system. (See Using SetData with Blank Cells below for more information on working with blanks.)

Keep in mind that a single row or single column of data is still represented as a two-dimensional array. The only difference in this case is that the size of one of the array bounds is one. When the size of both array bounds are one, then the array basically represents a single cell. Only scalars in Arius can contain single cells, and this property can also be used to read and write values to these objects as well.

Return:

Variant containing a two-dimensional array of numbers or empty strings (when property is being read).

Applies to:

*.apj files

VBA Examples:

See the following pages for several specific examples.

NOTE: For Scalar objects set TableType=”Scaler”.

Pull triangle data from Arius to Excel

Assume the following data is in an Excel spreadsheet:

FIGURE 1

	A	B	C	D	E	F	G	H
1	Accident Year	12	24	36	48	60	72	84
2	Prior	175	425	600	670	700	725	750
3	2008	200	450	650	730	780	805	
4	2009	200	400	550	600	675		
5	2010	230	520	640	700			
6	2011	290	515	690				
7	2012	270	570					
8	2013	300						

Also assume the following code is contained within a module in the workbook:

```
Dim AriusProject As New AriusSystem

Function GetData(Filename As String, SegmentName As String, TableType As String, TableRef As Variant,
Optional PartRef As Variant)

    On Error Resume Next
    GetData = AriusProject.Data(Filename, SegmentName, TableType, TableRef, PartRef)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

If you wanted the data in the sheet to come from Arius, you could use the following array formula in the range B2:H8:

```
{=GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL",
"Input", "Paid Loss")}
```

If you also wanted the labels to come from the Arius file, then you can use the array formulas below assuming you also have the following additional code in your workbook.

```
Function GetRowLabels(Filename As String, SegmentName As String, TableType As String, TableRef As Variant,
Optional PartRef As Variant)

    On Error Resume Next
    GetRowLabels = AriusProject.RowLabels(Filename, SegmentName, TableType, TableRef, PartRef)
    If Err.Number <> 0 Then

MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

```
Function GetColumnLabels(Filename As String, SegmentName As String, TableType As String, TableRef As
Variant, Optional PartRef As Variant)

    On Error Resume Next
    GetColumnLabels = AriusProject.ColumnLabels(Filename, SegmentName, TableType, TableRef, PartRef)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

RANGE	ARRAY FORMULA
B1:H1	{ =GetColumnLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Exhibit", "Paid Loss Development")}
A2:A7	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\Arius\ DemoFiles\Arius_Sample.APJ", “GL”, "Exhibit", "Paid Loss Development"))}

Notice that we need to use the Excel TRANSPOSE function on GetRowLabels. The reason for this is because GetColumnLabels and GetRowLabels return a one-dimensional array as opposed to GetData which returns a two dimensional array. Values in a one-dimensional array flow across the sheet (horizontally). In the case of column labels, this is desired. For row labels, however, we want the labels to flow down the sheet (vertically). In order to do this, we must use the TRANSPOSE function.

Push data from Excel to Arius

Refer back to Figure 1 and now assume that the data came from somewhere else (perhaps you manually typed it into the worksheet), and you wanted to set this data into an Arius file, you could call the following macro to write the data to the Paid Loss input triangle in the specified segment:

```
Function SetAriusData()

    Filename = "C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ"
    AriusProject.Data(Filename, “GL”, "Input", "Paid Loss") = Range("B2", "H8").Value
    AriusProject.WriteFile(Filename)

End Function
```

Note that the actual range for the data on the spreadsheet is B2:H8. Do not include any row or column labels. You cannot write these back to the file as these are read-only properties determined by the file structure. You must also call the WriteFile method at some point after setting data otherwise your changes will be lost. For efficiency, all changes are made to the copy of the file in memory.

Recreate an Arius Exhibit in Excel

Now let's assume you wanted to recreate an Arius exhibit in Excel (see Figure 2).

FIGURE 2

	A	B	C	D	E	F	G	H
1	Accident Year	12-24	24-36	36-48	48-60	60-72	72-84	84-Ult
2	Prior	2.429	1.412	1.117	1.045	1.036	1.034	
3	2008	2.250	1.444	1.123	1.068	1.032		
4	2009	2.000	1.375	1.091	1.125			
5	2010	2.261	1.231	1.094				
6	2011	1.776	1.340					
7	2012	2.111						
8								
9	Average	2.138	1.360	1.106	1.079	1.034	1.034	
10	Average Excluding High/Low	2.155	1.376	1.105	1.068			
11	Volume Weighted Average	2.110	1.355	1.107	1.078	1.034	1.034	
12	Time Weighted Average	2.072	1.337	1.101	1.093	1.033	1.034	
13	3 Year Average	2.049	1.315	1.103	1.079	1.034	1.034	
14								
15	Inverse Power Curve	2.188	1.286	1.124	1.069	1.043	1.030	1.102
16	Exponential Curve	1.728	1.360	1.178	1.088	1.043	1.021	1.021
17	Weibull Curve	2.011	1.352	1.158	1.077	1.040	1.021	1.024
18								
19	Selected	2.110	1.350	1.105	1.080	1.034	1.034	1.074

Use the following array formulas to create the exhibit in Excel as shown in Figure 2. Assume you have the GetColumnLabels, GetRowLabels, and GetData functions defined correctly. You will also need to format the sheet. The API currently does not handle any formatting. You would need to either manually format the data yourself or utilize VBA macros. A discussion on how to format data is beyond the scope of this document.

RANGE	ARRAY FORMULA
B1:H1	{ =GetColumnLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL", "Exhibit", "Paid Loss Development")}
A2:A7	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL", "Exhibit", "Paid Loss Development"))}
B2:H7	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL", "Exhibit", "Paid Loss Development")}
B9:H13	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL", "Exhibit", "Paid Loss Development","Statistics")}
A9:A13	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "GL", "Exhibit", "Paid Loss Development","Statistics"))}

RANGE	ARRAY FORMULA
B15:H17	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Exhibit", "Paid Loss Development","TFA")}
A15:A17	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\ Arius\DemoFiles\Arius_Sample.APJ", “GL”, "Exhibit", "Paid Loss Development","TFA"))}
B19:H19	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Exhibit", "Paid Loss Development","Selected")}

Duplicate an Arius Method in Excel

FIGURE 3

	A	B	C	D	E
1	Accident Year	Cumulative Paid Loss	Selected Development Factors	Cumulative Development Factors	Ultimate Loss
2	Prior	$750	1.074	1.074	$806
3	2008	$805	1.034	1.111	$894
4	2009	$675	1.034	1.148	$775
5	2010	$700	1.080	1.240	$868
6	2011	$690	1.105	1.370	$946
7	2012	$570	1.350	1.850	$1,054
8	2013	$300	2.110	3.903	$1,171

This method can be created in Excel in three different ways. The most direct way is to retrieve the data (excluding labels) in one formula array. We would also like to get the labels as well. Use the following formula arrays to accomplish this. Remember that you will also need to format the sheet in order to resemble Figure 3.

RANGE	ARRAY FORMULA
B1:E1	{ =GetColumnLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development")}
A2:A8	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\Arius\ DemoFiles\Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development"))}
B2:E8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development")}

We can also retrieve each column individually using the following arrays instead:

RANGE	ARRAY FORMULA
B1:E1	{ =GetColumnLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles \Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development")}
A2:A8	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\Arius\ DemoFiles\Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development"))}
B2:B8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", 1)}
C2:C8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", 2)}
D2:D8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", 3)}
E2:E8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", 4)}

You should notice here that we are getting each column one at a time by specifying the column number using the PartRef parameter for GetData. We can specify which column to retrieve if we are only interested in a subset of the method or report. In the example above, however, it is better to get the data all at once.

In addition, we can also use the following formula arrays to arrive at the same result on the sheet:

RANGE	ARRAY FORMULA
B1:E1	{ =GetColumnLabels("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development")}
A2:A8	{ =TRANSPOSE(GetRowLabels("C:\Users\Your_Username\Documents\Milliman\Arius\ DemoFiles\Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development"))}
B2:B8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", "Cumulative Paid Loss")}
C2:C8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\

RANGE	ARRAY FORMULA
	Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", “Selected Development Factors”)}
D2:D8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", “Cumulative Development Factors”)}
E2:E8	{ =GetData("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\ Arius_Sample.APJ", “GL”, "Method", "Paid Loss Development", “Ultimate Loss”)}

You will notice in the formula arrays above that we are referring to each column of data by name rather than by number. The names are the same as what would be returned from GetColumnLabels. The above example is for illustrative purposes only. If you wanted to retrieve the entire method or report, you would normally use a single formula array to get all the data at once.

Use SetData with blank cells

The default behavior for SetData replaces the existing data in the Arius table with whatever is in the source data, even if the source data contains blank cells. Also, the API does not support setting data in individual cells; the entire table must be set at one time.

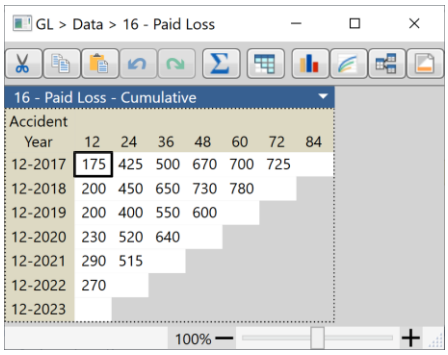
In certain situations, it may be more appropriate to ignore blank or empty cells in Excel when using the Data property to set data in an Arius file.

- For example, suppose you have a new diagonal of data in Excel as shown in Figure 4 below, and you want to set it into the Paid Loss triangle in Arius as the latest diagonal (see Figure 5).
- The default action of SetData would erase any data not in the latest diagonal in the Arius Paid Loss table.

FIGURE 4

	A	B	C	D	E	F	G	H
1	Accident Year	12	24	36	48	60	72	84
2	Prior							750
3	2008						805	
4	2009					675		
5	2010				700			
6	2011			690				
7	2012		570					
8	2013	300						

FIGURE 5



Arius does, however, provide a way to add the new (Excel) diagonal to the existing (Arius) triangle. The IgnoreBlanks pseudo PartRef of the Data property makes the API ignore the blank cells in the Excel source array and only set the cells containing data.

You can use IgnoreBlanks in one of the three ways below, depending on whether the data should be interpreted as cumulative or incremental.

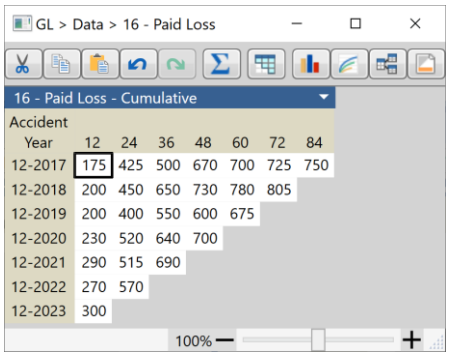
```
AriusProject.Data(Filename, “GL”, “Input”, “Paid Loss”, “Cumulative,IgnoreBlanks”) = Range(“B2”, “H8”).Value

AriusProject.Data(Filename, “GL”, “Input”, “Paid Loss”, “Incremental,IgnoreBlanks”) = Range(“B2”, “H8”).Value

AriusProject.Data(Filename, “GL”, “Input”, “Paid Loss”, “IgnoreBlanks”) = Range(“B2”, “H8”).Value
```

When IgnoreBlanks is used by itself, the data is assumed to be incremental. Since the example is setting cumulative data, we would use the first statement above. The resulting Paid Loss input triangle would look like the one in Figure 6.

FIGURE 6



It is still important to call WriteFile after changing data to ensure that the Arius file is correctly updated. The Arius file you are updating must not be open in Arius at the time, or the call to WriteFile may fail.

DeleteSegment

Syntax:

Sub DeleteSegment(filename As String, SegmentName As String)

Description:

Deletes the specified segment in the specified file.

Return:

Deletes segment

Applies to:

*.apj files

VBA example:

```
Sub DeleteSegment(Filename As String, SegmentName As String)

On Error GoTo errHandler
AriusProject.DeleteSegment(Filename, segmentName)
Exit Sub

errHandler:
MsgBox "There was an error deleting the specified segment from the file" & Filename
Resume Next
End Sub
```

DevelopmentPeriodLength

Syntax:

Property DevelopmentPeriodLength(filename As String) As Long

Description:

Read only. Gets the development period length from the specified file.

Return:

Long

1 = Months
3 = Quarters
6 = Half-years
12 = Years

Applies to:

*.apj files

VBA example:

```
Function GetDevelopmentPeriodLength(filename As String) As Long

    On Error Resume Next
    GetDevelopmentPeriodLength = AriusProject.DevelopmentPeriodLength(filename)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

Usage:

```
=
GetDevelopmentPeriodLength("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\
Sample.APJ")
```

returns the development period length for filename.

ErrorCode

Syntax:

Property ErrorCode(type As ErrorType) As Long

Description:

Read only. Returns the exact error code for a particular error type. This is useful when you want to check for a particular kind of error.

Return:

Long

Applies to:

*.apj files

VBA example:

```
Sub SetCompanyDescription(Filename As String, Description As String)

    On Error Resume Next
    AriusProject.Description(Filename) = Description
    If Err.Number = AriusProject.ErrorCode(E_FILE_NOT_RECOGNIZED) Then

        Handle when file not recognized for the particular API call

    End If

End Sub
```

ErrorMessage

Syntax:

Property ErrorMessage(ErrNum As Long) As String

Description:

Read only. Returns a formatted error message based on the error code supplied as a parameter.

Return:

String

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem
On Error Resume Next
' Make a call to the API
If Err.Number <> 0 Then
    MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
End If
```

ExposureDate

Syntax:

Property ExposureDate(filename As String) As Date

Description:

Read only. Retrieves the date of the first exposure period from the Arius file. This date is found in PROJECT SETTINGS | DATA STRUCTURE | DATE PARAMETERS.

Return:

Date

Applies to:

*.apjJ files

VBA example:

```
Function GetExposureDate(Filename As String)
    On Error Resume Next
    GetExposureDate = AriusProject.ExposureDate(Filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Entering the following as a single cell Excel array formula:

```
= GetExposureDate("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\
Arius_Sample.APJ")
```

returns a date.

ExposurePeriodLength

Syntax:

Property ExposurePeriodLength(filename As String) As Long

Description:

Read only. Retrieves the length of the exposure periods in an Arius file.

Return:

Long

1 = Months
3 = Quarters
6 = Half-years
12 = Years

Applies to:

*.apj files

VBA example:

```
Function GetExposurePeriodLength(filename As String) As Long
    On Error Resume Next
        GetExposurePeriodLength = AriusProject.ExposurePeriodLength(filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Entering the following formula into a cell:

```
=GetExposurePeriodLength("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\
Arius_Sample.APJ")
```

returns the 'length of the exposure period' in filename.

FirstDevelopmentMonth

Syntax:

Property FirstDevelopmentMonth(filename As String) As Long

Description:

Read only. Returns the month number of the first development period.

Return:

Long

Applies to:

*.apj files

VBA example:

```
Function GetFirstDevelopmentMonth(Filename As String)
    On Error Resume Next
    GetFirstDevelopmentMonth = AriusProject.FirstDevelopmentMonth(Filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Entering the following formula into a cell:

```
= GetFirstDevelopmentMonth("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\
SomeFile.APJ")
```

returns the first development month for the Arius file.

FirstExposureYear

Syntax:

Property FirstExposureYear(filename As String) As Long

Description:

Read only. Returns the year of the first exposure period.

Return:

Long

Applies to:

*.apj files

VBA example:

```
Function GetFirstExposureYear(filename As String) As Integer

    On Error Resume Next
    GetFirstExposureYear = AriusProject.FirstExposureYear(filename)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

Usage:

Entering the following formula into a cell:

```
=GetFirstExposureYear("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\
Arius_Sample.APJ")
```

returns the 'year of the first exposure period' in filename.

FlushFile

Syntax:

Sub FlushFile(filename As String)

Description:

This method purges the specified file from memory only and does not delete or modify the file on the disk.

Return:

Nothing is returned.

Applies to:

Any file type in memory.

VBA example:

```
Sub FlushFile(Filename As String)

On Error GoTo errHandler

    AriusProject.FlushFile Filename
    Exit Sub

errHandler:

    MsgBox "There was an error removing the cached file " & Filename
    Resume Next

End Sub
```

HasPrior

Syntax:

Property HasPrior(filename As String) As Long

Description:

Read only. Determines whether first exposure period includes All Prior row. Returns zero if no prior row, otherwise a nonzero value indicates there is a prior row.

Return:

Long (0 = false, 1 = true)

Applies to:

*.apj files

VBA example:

```
Function GetHasPrior(filename As String)
    If AriusProject.HasPrior(filename) Then
        GetHasPrior = TRUE
    Else
        GetHasPrior = FALSE
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then:

```
=GetHasPrior(filename)
```

returns TRUE or FALSE based upon the presence of a prior row in filename, and the wrapper function as defined in example above.

NumColumns

Syntax:

Property NumColumns(filename As String, SegmentName As String, TableType As String, TableRef, [PartRef]) As Integer

Description:

Read only. Returns the number of columns for any table in a particular file. Will also return the number of columns for the subpart specified for a particular table (e.g., Statistics).

Return:

Integer

Applies to:

*.apj files

VBA example:

```
Function GetNumColumns(Filename As String, SegmentName As String, TableType As String, TableRef As
Variant, Optional PartRef As Variant) As Integer

    On Error Resume Next
    GetNumColumns = AriusProject.NumColumns(Filename, SegmentName, TableType, TableRef, PartRef)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula. Enter the following as a single cell Excel array formula:

- This formula returns an integer with the number of columns in the Paid Loss input table for the SegmentName in filename.

  ```
  = GetNumColumns(filename, "GL", "input", "Paid Loss")
  ```

- This formula returns an integer with the number of columns in the statistics block within the Paid Loss Development exhibit for the SegmentName in filename.

  ```
  =GetNumColumns(filename, " SegmentName ", "exhibit", "Paid Loss Development", "statistics")
  ```

NumDevelopments

Syntax:

Property NumDevelopments(filename As String) As Long

Description:

Read only. Returns the number of development periods (columns) specified for a particular Arius file.

Return:

Long

Applies to:

*.apj files

VBA example:

```
Function GetNumDevelopments(filename As String)
    On Error Resume Next
    GetNumDevelopments = AriusProject.NumDevelopments(filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then:

```
= GetNumDevelopments(filename)
```

returns the number of development periods in filename.

NumExposures

Syntax:

Property NumExposures(filename As String) As Long

Description:

Read only. Returns the number of exposure periods (rows) specified in an Arius file.

Return:

Long

Applies to:

*.apj files

VBA example:

```
Function GetNumExposures(filename As String) As Integer
    On Error Resume Next
    GetNumExposures = AriusProject.NumExposures(filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then:

```
=GetNumExposures(filename)
```

returns the 'number of exposure periods' in filename.

NumRows

Syntax:

Property NumRows(filename As String, SegmentName As String, TableType As String, TableRef, [PartRef]) As Integer

Description:

Read only. Returns the number of rows for the specified table in an Arius file. When PartName is supplied, will return the number of rows for that subpart of the specified table (e.g., Statistics).

Return:

Integer

Applies to:

*.apj files

VBA example:

```
Function GetNumRows(Filename As String, SegmentName As String, TableType As String, TableName As String)
    On Error Resume Next
    GetNumRows = AriusProject.NumRows(Filename, SegmentName, TableType, TableName)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula to an Arius file, then entering the following as a single cell Excel array formula:

```
= GetNumRows(filename, "GL", "input", "Paid Loss")
```

returns an integer with the number of rows in the Paid Loss input table for the SegmentName in filename.

OpenFile

Syntax:

Sub OpenFile(filename As String)

Description:

Opens the specified file and reads contents into memory. Be sure to call CloseFile when done.

Return:

Nothing is returned.

Applies to:

*.apj files

VBA example:

```
Sub OpenFile(Filename As String)

On Error GoTo errHandler

    AriusProject.OpenFile Filename
    Exit Sub


errHandler:

    MsgBox "There was an error opening the file " & Filename
    Resume Next

End Sub
```

PeriodType

Syntax:

Property PeriodType(filename As String) As String

Description:

Read only. Returns the text stored in the Arius file for PROJECT SETTINGS | DATA STRUCTURE | EXPOSURE PERIOD TYPE.

Return:

String

Applies to:

*.apj files

VBA example:

```
Function GetPeriodType(filename As String)
    On Error Resume Next
    GetPeriodType = AriusProject.PeriodType(filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula to an Arius file, then:

```
= GetPeriodType(filename)
```

returns the text for Period Type for filename.

RecalcSegments

Syntax:

Sub RecalcSegments(filename as a string)

Description:

This method will refresh calculations in segments where the project identified by filename is using calculated segments. This is intended to be used where data is updated through the API in data tables which are the basis of calculated objects in calculated segments. This refreshes calculated segment objects in the Arius project which is temporarily loaded into memory for use by the API only. It is still important to call WriteFile after recalculating segments to ensure that the Arius file on disk is correctly updated.

VBA example:

```
Sub RecalcSegments(Filename As String)

On Error GoTo errHandler

    AriusProject.RecalcSegments Filename
    Exit Sub


errHandler:

    MsgBox "There was an error recalculating segments for  " & Filename
    Resume Next

End Sub
```

RowLabels

Syntax:

Property RowLabels(filename As String, SegmentName As String, TableType As String, TableRef, [PartName]) As Variant

Description:

Read only. Returns a string array with the row labels for a particular table in an Arius file. When PartName is supplied, it will return the row labels for that subpart of the table (e.g., Exhibit Statistics).

Return:

Variant containing an array of strings

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem
Function GetRowLabels(Filename As String, SegmentName As String, TableType As String, TableName As String)
    On Error Resume Next
    GetRowLabels = AriusProject.RowLabels(Filename, SegmentName, TableType, TableName)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then entering the following as an Excel array formula:

```
= GetRowLabels(filename, "GL", "input", "Paid Loss")
```

returns a string array with the row labels for the Paid Loss triangle.

Alternatively, you can modify this function to include the PartName parameter to return the names of the different parts of an exhibit. For example:

```
= GetRowLabels(filename, "GL", "exhibit", "Paid Loss Development", "statistics")
```

would effectively return the names of the statistics in the Paid Loss Development exhibit.

RunService

Syntax:

Sub RunService(filename As String, [SegmentName As String], ServiceName As String)

*SegmentName is optional

Description:

This method runs the specified service for the specified segment name found in the specified filename. If the SegmentName string is empty, the specified ServiceName is run for all segments within the file.

The available services include:

Diagnostics

Simulation

SuggestedHetero

Return:

Nothing is returned.

Applies to:

*.apj files

VBA example:

```
Sub RunService(filename As String, [SegmentName As String], ServiceName As String)
On Error GoTo errHandler

    AriusProject.RunService Filename, SegmentName, Diagnostics
    Exit Sub


errHandler:

    MsgBox "There was an error running the specified service for the cached file " & Filename
    Resume Next

End Sub
```

SaveFileAs

Syntax:

Sub SaveFileAs(SourceFileName As String, DestinationFileName As String)

Description:

Makes a copy of the source file. The source file does not need to be previously opened or loaded into memory. Calling SaveFile will rename the file in memory so only one file will exist (as opposed to two). You will need to call FlushFile on the renamed file to remove it from memory.

Warning:

This routine will use the .apj file format to write any data to disk and thus will generate an error message if the destination file does not have an .apj extension.

Return:

Nothing is returned.

Applies to:

*.apj files

VBA example:

```
Sub SaveFileAs(SourceFileName As String DestinationFileName As String)

On Error GoTo errHandler

    AriusProject.SaveFileAs SourceFileName, DestinationFileName
    Exit Sub


errHandler:

    MsgBox "There was an error saving the file " & Filename1 & " with the name " & Filename2
    Resume Next

End Sub
```

ScalingFactor

Syntax:

Property ScalingFactor(filename As String) As Double

Description:

Read only. Returns the scaling factor assigned inside an Arius file.

Return:

Double

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem
Function GetScalingFactor(filename As String)

    On Error Resume Next
    GetScalingFactor = AriusProject.ScalingFactor(filename)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then entering the following as a single cell Excel formula:

```
= GetScalingFactor(filename)
```

returns the scaling factor used in filename.

Segments

Syntax:

Sub Segments(Filename As String)

Description:

Read-only. Returns the text representing all the segments for the given Filename.

Return:

Variant containing an Array of Strings

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem

Function GetSegments(Filename As String)

    On Error Resume Next
    GetSegments = AriusProject.Segments(Filename)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

TableIndexes

Syntax:

Property TableIndexes(filename As String, TableType As String) As Variant

Description:

Read only. Returns an array of integers containing all the indexes for a particular table type in the file specified. Any of the values returned can then be used for the TableRef parameter in other API calls such as RowLabels where TableRef is required. The list returned is sorted low to high.

Return:

Variant containing an array of integers

Applies to:

*.apj files

VBA example:

```
Function GetTableIndexes(filename As String, TableType As String) As Variant

    On Error Resume Next
    GetTableIndexes = AriusProject.TableIndexes(filename, TableType)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If
    End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then entering the following as an Excel array formula:

```
= GetTableIndexes(filename,"Input" )
```

returns an array of integers containing all of the table IDs for all the Input tables.

TableInfo

Syntax:

Property TableInfo(filename As String, TableType As String) As Variant

Description:

Returns a two-dimensional array containing information on the table names and indexes in the particular file. The first column of the array contains the table indexes. The second column contains the corresponding table's long name. And the third column contains the abbreviated or short name if one is available. The TableType can be Input, Exhibit, Method, or Report.

Return:

Variant containing a two-dimensional array of strings and numbers
Applies to:

*.apj files

VBA example:

```
Function GetTableInfo(filename As String, TableType As String)

    On Error Resume Next
    GetTableInfo = AriusProject.TableInfo(filename, TableType)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then entering the following as an Excel array formula:

```
= GetTableInfo(filename,"Input")
```

returns a two dimensional array containing IDs and names of all the input tables.

TableNames

Syntax:

Property TableNames(filename As String, TableType As String) As Variant

Description:

Read only. Returns an array of strings representing all the names of all tables for a particular table type. The arrays are returned in alphabetical order.

Return:

Variant containing an Array of Strings

Applies to:

*.apj files

VBA example:

```
Function GetTableNames(filename As String, TableType As String) As Variant
    On Error Resume Next
    GetTableNames = AriusProject.TableNames(filename, TableType)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

Usage:

Suppose filename is a cell's range name, and the cell contains a fully qualified path name string or formula, then entering the following as an Excel array formula:

```
=GetTableNames(filename,"Input" )
```

returns an array of strings containing all the table names for all the Input tables.

TableParts

Syntax:

Sub TableParts(Filename As String, TableType As String, TableName As String)

Description:

Read-only. Returns the text representing the various parts table parts of the specified table type of the specified table name of the specified Filename.

Return:

Variant containing an array of strings

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem

Function GetTableParts(Filename As String, TableType As String, TableName As String)
    On Error Resume Next
    GetTableParts = AriusProject.TableParts(Filename, TableType, TableName)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function
```

TableTypes

Syntax:

Property TableTypes(filename As String) As Variant

Description:

Read only. Returns an array of strings representing each of the table types for a particular file.

Return:

Variant containing an array of strings. The table types listed here represent those used within the Arius Deterministic module. All other table types returned by this function are limited in use to the Arius Stochastic module (for example “Correlation,” “ODP,” “Aggr,” etc.)

“Input”

“Exhibit”

“Method”

“Report”

“Scalar”

Applies to:

*.apj files

VBA example:

```
Dim AriusProject as New AriusSystem
Function GetTableTypes(Filename As String)

    On Error Resume Next
    GetTableTypes = AriusProject.TableTypes(Filename)
    If Err.Number <> 0 Then

        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"

    End If

End Function
```

Usage:

Suppose filename is a cell’s range name, and the cell contains a fully qualified path name string or formula, then entering the following as an Excel array formula:

```
= GetTableTypes(filename)
```

returns an array of strings containing all the hard coded table types.

WriteFile

Syntax:

Sub WriteFile(filename As String)

Description:

Writes the contents of the file in memory to the disk. Overwrites what was there previously. WriteFile will automatically call OpenFile and CloseFile if necessary (assuming the file is already loaded in memory).

Warning:

This routine will use the .apj file format to write any data to disk and thus will generate an error message if the destination file does not have an .apj extension.

Return:

Nothing is returned.

Applies to:

*.apj files

VBA example:

```
Sub WriteFile(Filename As String)

On Error GoTo errHandler

    AriusProject.WriteFile Filename
    Exit Sub


errHandler:

    MsgBox "There was an error writing the file " & Filename
    Resume Next

End Sub
```

ValDate

Syntax:

Property ValDate(filename As String) as String

Description:

Read Only. Retrieves the As of Date for specified Arius filename.

Return:

The API will return a string whose contents are the evaluation date of the Arius file, formatted according to your local settings.

Applies to:

*.apj files

VBA example:

```
Sub GetAsOfDate(filename As String) as String

On Error GoTo errHandler

    GetAsOfDate = AriusProject.ValDate(filename)
    Exit Sub

errHandler:

    MsgBox "There was an error retrieving the As Of Date from the file " & filename
    Resume Next

End Sub
```