Arius API
User Guide

An overview of how to use the exposed Arius functions within Microsoft Excel VBA
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.
1301 Fifth Avenue, Suite 3800
Seattle WA 98101-2605

Milliman.com

© Copyright 2006-2018 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 .................................................................................................................. 1
2. BEFORE YOU GET STARTED WITH THE API .................................................................................. 2
3. EXAMPLE EXCEL WORKBOOKS ....................................................................................................... 3
   Example 1: Batch Get Data .................................................................................................................. 5
   Example 2: Specific Get Data ............................................................................................................. 6
   Example 3: Get File Properties .......................................................................................................... 7
   Example 4: Set Data ............................................................................................................................ 8
   Example 4B: Get and Set Data (includes VBA tutorial) ...................................................................... 9
      Using the Example 4B workbook to retrieve or write data to/from Arius ........................................ 9
      Updating only the last diagonal in Arius ........................................................................................... 10
   Using this workbook as a VBA tutorial ............................................................................................... 10
   Example 4B (Variation): Get and Set Data with RecalcSegment ....................................................... 11
   Example 5: Summary .......................................................................................................................... 12
   Example 6: Send triangles in external file to Arius ........................................................................... 13
   Example 7: Update triangles in Arius .................................................................................................. 14
   Example 8: Automate TriangleMaker to send input triangles to Arius ............................................. 15
      TMAutomation module .................................................................................................................... 17
      SetAriusData module ....................................................................................................................... 18
      Main module ..................................................................................................................................... 18
   Example 9: Automate TriangleMaker & update Input triangles in Arius ........................................... 19
   Example 10: Run Arius Stochastic models ......................................................................................... 20
   Example 11: Batch Get Data by type: Data, Exhibit, Method, and Report ......................................... 22
4. THE ROOT OF ALL API CALLS: THE ARIUSSYSTEM OBJECT ....................................................... 24
   Create an instance of the Arius System Object (AriusSystem) ......................................................... 24
   Global scope instances vs. local scope instances of AriusSystem .................................................... 24
5. OPENING AND CLOSING FILES WITH FLUSHFILE ................................................................... 25
6. CONSIDERATIONS WHEN NAMING SEGMENTS ...................................................................... 26
7. PROPERTIES VS. METHODS .......................................................................................................... 27
8. CHECKING FOR ERRORS ............................................................................................................... 28
9. SUMMARY LIST OF API METHODS AND PROPERTIES ............................................................. 29
   Methods ................................................................................................................................................ 29
   Properties ............................................................................................................................................ 30
10. API DETAILS .................................................................................................................................. 33
    Parameters of Arius API methods & properties ................................................................................ 33
        Filename as string ............................................................................................................................ 33
        TableType as string ........................................................................................................................ 33
        TableRef as variant .......................................................................................................................... 33
        [PartRef] as variant .......................................................................................................................... 33
    Example code using subroutines and functions .............................................................................. 35
    AsOfDate ........................................................................................................................................... 36
CloseFile ............................................................................................................................ 37
ColumnLabels .................................................................................................................. 38
Comment ......................................................................................................................... 39
CopySegment .................................................................................................................. 40
Data .................................................................................................................................... 41
  Pull triangle data from Arius to Excel .......................................................................... 42
  Push data from Excel to Arius ..................................................................................... 43
  Recreate an Arius Exhibit in Excel .............................................................................. 44
  Duplicate an Arius Method in Excel ............................................................................ 45
  Use SetData with blank cells ....................................................................................... 47
DeleteSegment ................................................................................................................ 49
DevelopmentPeriodLength ............................................................................................... 50
ErrorCode ....................................................................................................................... 51
ErrorMessage ............................................................................................................... 52
ExposureDate .................................................................................................................. 53
ExposurePeriodLength ..................................................................................................... 54
FirstDevelopmentMonth ................................................................................................. 55
FirstExposureYear .......................................................................................................... 56
FlushFile ......................................................................................................................... 57
HasPrior ......................................................................................................................... 58
NameLookup ................................................................................................................... 59
NumColumns .................................................................................................................. 60
NumDevelopments .......................................................................................................... 61
NumExposures ............................................................................................................... 62
NumRows ....................................................................................................................... 63
OpenFile ......................................................................................................................... 64
PeriodType ..................................................................................................................... 65
RecalcSegments ............................................................................................................. 66
RowLabels ..................................................................................................................... 67
RunService ..................................................................................................................... 68
SaveFileAs ...................................................................................................................... 69
ScalingFactor ................................................................................................................... 70
Segments ......................................................................................................................... 71
TableIndexes .................................................................................................................. 72
TableInfo ......................................................................................................................... 73
TableNames .................................................................................................................... 74
TableParts ....................................................................................................................... 75
TableTypes ..................................................................................................................... 76
WriteFile ......................................................................................................................... 77
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 (*.API) 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 2010, 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 My 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 of 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 was written to include a VBA and API

**NOTE:**

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

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). You are encouraged to 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).
EXAMPLE 2: SPECIFIC GET DATA

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.
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.
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 first 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 the first button. 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).
EXAMPLE 4B: GET AND SET DATA (INCLUDES VBA TUTORIAL)

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.

The workbook contains two tabs, INSTRUCTIONS – READ ME FIRST and SET PARAMETERS. This workbook demonstrates how to use the API to retrieve one, or many, input data tables from Arius and also send data back to an input 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 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 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.

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

On the Set Parameters tab you supply file names and paths, as well as the Segments and data object Table names that you wish to retrieve from Arius. Table names must match Arius exactly. Arius long or abbreviated names may be used. In addition you can specify cumulative or incremental data and a customized range name.

Note: Arius will not write to a calculated table.
To write data to Arius, your Arius project must be closed and the data you are writing must be the same size and shape as your Arius project.

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 or a unique default range name if a custom name is not specified in this grid. The parameters grid found on this tab has been assigned the named range DataRequirements through Excel. To list more or fewer tables for your batch Get Data and/or Set Data request, you can insert or delete rows from this grid and the code will function normally as long as the range name continues to define the light blue area of grid rows.

Pressing the Retrieve Data button will create a new tab for each segment and each table requested for this segment will be retrieved from Arius and will populate the segment tab. Unlike Exercise 4 after which this exercise is modeled, data values are displayed on the segment tab rather than array formulas. When worksheets are deleted, the range names remain so that external references to these range names will not be broken.

It is assumed that you will retrieve data tables from Arius to capture the current Arius structure (i.e., you will have appended your new diagonal before reading from Arius, if applicable). 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. Therefore, retrieving data from Arius as a first step will set-up the current Arius structure (row/column labels) and range names for your use. Column and Row headings are for reference only, and are ignored when writing to Arius.

Also, unlike Exercise 4, because the retrieved data is displayed as values, you can make changes to the data directly in the retrieved triangle (or column) and write this modified data to Arius. Please be careful to create a copy of your Arius project before using this workbook.

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 a triangle will write a zero to the Arius table.

Using this workbook as a VBA tutorial

The worksheet INSTRUCTIONS—READ ME FIRST 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 INSTRUCTIONS—READ ME FIRST worksheet.
EXAMPLE 4B (VARIATION): GET AND SET DATA WITH RECALCSEGMENT

See section above EXAMPLE 4B: GET AND SET DATA (INCLUDES VBA TUTORIAL). This variation of example 4B is identical except that the example 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 “Columns” 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).
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

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:
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.

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

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

```vba
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.**

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

**Subroutine ClearTMData**

**Purpose:**
Clears the contents and formats the “WorkArea” tab.

**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.API, 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.
  ```vba
  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.
  ```vba
  AriusProject.CopySegment AriusOutLocation & "\" & Filename, DummySegment
  AriusProject.WriteFile AriusOutLocation & "\" & Filename
  ```
- Then it will send the triangles to Arius.
  ```vba
  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.
    ```vba
    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

This workbook contains two tabs: SETUP and WORKAREA.

- It is 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 getting 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 hetero 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",

  AriusProject.Data(FolderName & "\" & NewFileName, Segment, "ODP Paid",

- 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.
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 data object names that you wish to retrieve from Arius. Object names must match Arius exactly. Arius long or abbreviated names may be used. In addition you can specify a customized range name. Range names will be assigned to the retrieved object’s data using the custom range names you provide or a unique default range name if a custom name is not specified in the grid. Each parameters grid has been assigned a named range through Excel. To list more or fewer tables in a grid, you can insert or delete rows from this 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 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 Data button will create a new tab for each segment with the tab name identifying the data type (Input, Method, Exhibit, 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 are displayed on the worksheets rather than array formulas. 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 project files. 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.

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

```plaintext
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)
```

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

```vba
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 Aria). 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)  
  - > (greater than)  
  - : (colon)  
  - = (double quote)  
  - / (forward slash)  
  - \ (backslash)  
  - | (vertical bar)  
  - ? (question mark)  
  - * (asterisk)  

- 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.

NOTE:
Segment names are case sensitive when using the API.
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 CompanyName as String

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

CompanyName = AriusProject.Company(“C:\Sample1.APJ”) 'example retrieving a property value

AriusProject.Company(“C:\Sample1.APJ”) = “New Company Name” '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_OPEN_FAILURE
E_DICTIONARY_LOAD_FAILURE
E_NAME_NOT_RECOGNIZED
E_TABLE_NOT_FOUND
E_TABLE_PART_NOT_RECOGNIZED
E_TABLE_TYPE_NOT_RECOGNIZED
E_DATA_SOURCE_NOT_RECOGNIZED
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 ShowAllErrorMessage()
    Dim etype(0 To 10) As ARIUSAPILib.ErrorType
    etype(0) = E_DATA_SOURCE_NOT_RECOGNIZED
    etype(1) = E_DICTIONARY_LOAD_FAILURE
    etype(2) = E_FILE_NOT_RECOGNIZED
    etype(3) = E_FILE_OPEN_FAILURE
    etype(4) = E_NAME_NOT_RECOGNIZED
    etype(5) = E_TABLE_NOT_FOUND
    etype(6) = E_TABLE_PART_NOT_RECOGNIZED
    etype(7) = E_TABLE_TYPE_NOT_RECOGNIZED
    etype(8) = E_UNKNOWN
    etype(9) = 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.

### METHODS

<table>
<thead>
<tr>
<th>METHOD NAME</th>
<th>PARAMETERS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseFile</td>
<td>(filename As String)</td>
<td>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.</td>
</tr>
<tr>
<td>CopySegment</td>
<td>(filename As String, SegmentName As String,</td>
<td>Makes a copy of the specified segment in the specified file.</td>
</tr>
<tr>
<td></td>
<td>NewSegmentName As String)</td>
<td></td>
</tr>
<tr>
<td>DeleteSegment</td>
<td>(filename As String, SegmentName As String)</td>
<td>Deletes the specified segment from the specified file.</td>
</tr>
<tr>
<td>FlushFile</td>
<td>(filename As String)</td>
<td>Purges the specified file from memory. Does not delete it from the disk.</td>
</tr>
<tr>
<td>OpenFile</td>
<td>(filename As String)</td>
<td>Opens the specified file and reads contents into memory. Be sure to call CloseFile when done.</td>
</tr>
<tr>
<td>RunService</td>
<td>(filename As String, [SegmentName As String],</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>ServiceName As String)</td>
<td></td>
</tr>
<tr>
<td>SaveFileAs</td>
<td>(filename1 As String, filename2 As String)</td>
<td>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.</td>
</tr>
<tr>
<td>WriteFile</td>
<td>(filename As String)</td>
<td>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).</td>
</tr>
</tbody>
</table>
### PROPERTIES

<table>
<thead>
<tr>
<th>PROPERTY NAME</th>
<th>PARAMETERS</th>
<th>RETURN TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsOfDate</td>
<td>(filename As String)</td>
<td>String</td>
<td>Sets and gets the As of Date for specified Arius filename. Since Arius uses a free form string for this field, formatting is not required or checked. The API will return whatever is there and you may use whatever format you like to set it.</td>
</tr>
<tr>
<td>ColumnLabels</td>
<td>(filename As String, SegmentName As String, TableType As String, TableRef, [PartName])</td>
<td>Array</td>
<td>Read only. Returns the text representing the column labels for the table or table subpart specified.</td>
</tr>
<tr>
<td>Comment</td>
<td>(filename As String)</td>
<td>String</td>
<td>Sets and gets the comment field for the Arius file.</td>
</tr>
<tr>
<td>Data</td>
<td>(filename As String, SegmentName As String, TableType As String, TableRef, [PartRef])</td>
<td>&lt;depends on request&gt;</td>
<td>Sets and gets specified data for specified object.</td>
</tr>
<tr>
<td>DevelopmentPeriodLength</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Gets the development period length from the specified file.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>(type As ErrorType)</td>
<td>Long</td>
<td>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.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>(ErrNum As Long)</td>
<td>String</td>
<td>Read only. Returns a formatted error message based on the error code supplied as a parameter.</td>
</tr>
<tr>
<td>ExposureDate</td>
<td>(filename As String)</td>
<td>Date</td>
<td>Read only. Retrieves the date of the first exposure period from the Arius file in PROJECT SETTINGS</td>
</tr>
<tr>
<td>ExposurePeriodLength</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Retrieves the length of the exposure periods in an Arius file.</td>
</tr>
<tr>
<td>PROPERTY NAME</td>
<td>PARAMETERS</td>
<td>RETURN TYPE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FirstDevelopmentMonth</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Returns the month number of the first development period.</td>
</tr>
<tr>
<td>FirstExposureYear</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Returns the year of the first exposure period.</td>
</tr>
<tr>
<td>HasPrior</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Returns zero if no prior row, otherwise a nonzero value indicates there is a prior row.</td>
</tr>
<tr>
<td>NameLookup</td>
<td>(filename As String, TableType As String, Index As Integer)</td>
<td>String</td>
<td>Read only. Returns the name of a table given its index and table type within the specified file.</td>
</tr>
<tr>
<td>NumColumns</td>
<td>(filename As String, SegmentName As String, TableType As String, TableRef, [PartRef])</td>
<td>Integer</td>
<td>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).</td>
</tr>
<tr>
<td>NumDevelopments</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Returns the number of development periods specified for a particular Arius file.</td>
</tr>
<tr>
<td>NumExposures</td>
<td>(filename As String)</td>
<td>Long</td>
<td>Read only. Returns the number of exposure periods specified in an Arius file.</td>
</tr>
<tr>
<td>NumRows</td>
<td>(filename As String, SegmentName As String, TableType As String, TableRef, [PartName])</td>
<td>Integer</td>
<td>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).</td>
</tr>
<tr>
<td>PeriodType</td>
<td>(filename As String)</td>
<td>String</td>
<td>Read only. Returns whatever information is stored in PROJECT SETTINGS</td>
</tr>
<tr>
<td>RowLabels</td>
<td>(filename As String, SegmentName As String, TableType As String, TableRef, [PartName])</td>
<td>Array of Strings</td>
<td>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).</td>
</tr>
<tr>
<td>PROPERTY NAME</td>
<td>PARAMETERS</td>
<td>RETURN TYPE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------</td>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Scaling Factor</td>
<td>(filename As String)</td>
<td>String</td>
<td>Read only. Returns the scaling factor assigned inside an Arius file.</td>
</tr>
<tr>
<td>Segments</td>
<td>(filename As String)</td>
<td>Array of Strings</td>
<td>Read only. Returns a string array with the segments in an Arius file.</td>
</tr>
<tr>
<td>TableIndexes</td>
<td>(filename As String, TableType As String)</td>
<td>Array of Integers</td>
<td>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.</td>
</tr>
<tr>
<td>TableNames</td>
<td>(filename As String, TableType As String)</td>
<td>Array of Strings</td>
<td>Read only. Returns an array of strings representing all the names of all tables for a particular table type.</td>
</tr>
<tr>
<td>TableParts</td>
<td>(filename As String, TableType As String, TableName As String)</td>
<td>Array of Strings</td>
<td>Read only. Returns an array of strings representing all of the table parts of the specified table type for the specified table name.</td>
</tr>
<tr>
<td>TableInfo</td>
<td>(filename As String, TableType As String)</td>
<td>Array of Integers and Strings</td>
<td>Read only. Returns a two-dimensional array containing information on the table names and indexes in the particular file.</td>
</tr>
<tr>
<td>TableTypes</td>
<td>(filename As String)</td>
<td>Array of Strings</td>
<td>Read only. Returns an array of strings representing each of the table types.</td>
</tr>
</tbody>
</table>
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 in to 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.
AsOfDate

Syntax:

Property AsOfDate(filename As String) As String

Description:

Sets and gets the As of Date for specified Arius filename. Since Arius uses a free form string for this field, formatting is not required or checked.

Return:

The API will return a string whose contents are whatever is in the Arius file using whatever format that was entered in there and you may use whatever format you like to set it.

Applies to:

*.APJ files

VBA example:

```vba
Sub GetAsOfDate(filename As String) As String
    On Error GoTo errHandler
    GetAsOfDate = AriusProject.AsOfDate(filename)
    Exit Sub

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

Sub SetAsOfDate(filename As String, NewAsOfDate As String)
    On Error GoTo errHandler
    AriusProject.AsOfDate(filename) = NewAsOfDate
    Exit Sub

errHandler:
    MsgBox "There was an error setting the As Of Date for the file " & filename
    Resume Next
End Sub
```
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 "GL" and "Closed Claim Development."
Comment

Syntax:
Property Comment(filename As String) As String

Description:
Sets and gets the comment field for the Arius file.

Return:
String

Applies to:
*.APJ files

VBA example:

Sub GetFileComment(filename As String) As String

On Error Resume Next
    GetFileComment = AriusProject.Comment(filename)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
        End If
Exit Sub

Sub SetFileComment(filename As String, NewComment As String)

On Error Resume Next
    AriusProject.Comment(filename) = NewComment
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
        End If
Exit Sub
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

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:

![Excel Spreadsheet](image)

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

```vba
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:

```excel
{=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.

```vba
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

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.

<table>
<thead>
<tr>
<th>RANGE</th>
<th>ARRAY FORMULA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1:H1</td>
<td>=GetColumnLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Exhibit&quot;, &quot;Paid Loss Development&quot;)</td>
</tr>
<tr>
<td>A2:A7</td>
<td>=TRANSPOSE(GetRowLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Exhibit&quot;, &quot;Paid Loss Development&quot;))</td>
</tr>
<tr>
<td>B2:H7</td>
<td>=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Exhibit&quot;, &quot;Paid Loss Development&quot;)</td>
</tr>
<tr>
<td>B9:H13</td>
<td>=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Exhibit&quot;, &quot;Paid Loss Development&quot;,&quot;Statistics&quot;)</td>
</tr>
<tr>
<td>A9:A13</td>
<td>=TRANSPOSE(GetRowLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Exhibit&quot;, &quot;Paid Loss Development&quot;,&quot;Statistics&quot;))</td>
</tr>
</tbody>
</table>
Duplicate an Arius Method in Excel

FIGURE 3

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

<table>
<thead>
<tr>
<th>RANGE</th>
<th>ARRAY FORMULA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1:E1</td>
<td>{=GetColumnLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;)}</td>
</tr>
<tr>
<td>A2:A8</td>
<td>{=TRANSPOSE(GetRowLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;))}</td>
</tr>
<tr>
<td>B2:B8</td>
<td>{=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;, 1)}</td>
</tr>
<tr>
<td>C2:C8</td>
<td>{=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;, 2)}</td>
</tr>
<tr>
<td>D2:D8</td>
<td>{=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;, 3)}</td>
</tr>
<tr>
<td>E2:E8</td>
<td>{=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;, 4)}</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>RANGE</th>
<th>ARRAY FORMULA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1:E1</td>
<td>{=GetColumnLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;)}</td>
</tr>
<tr>
<td>A2:A8</td>
<td>{=TRANSPOSE(GetRowLabels(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;))}</td>
</tr>
<tr>
<td>B2:B8</td>
<td>{=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;, &quot;Cumulative Paid Loss&quot;)}</td>
</tr>
<tr>
<td>C2:C8</td>
<td>{=GetData(&quot;C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ&quot;, &quot;GL&quot;, &quot;Method&quot;, &quot;Paid Loss Development&quot;, &quot;Cumulative Paid Loss&quot;)}</td>
</tr>
</tbody>
</table>
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**
Arius does, however, provide a way to add the new XLS diagonal to the existing .APJ 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.

```
```

```
```

```
```

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.

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:**

```vba
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:**

```vba
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:
*.APJ 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:**

```vba
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:

```vba
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.
NameLookup

Syntax:
Property NameLookup(filename As String, TableType As String, Index As Integer) As String

Description:
Read only. Returns the name of a table given its index and table type within the specified file.

Return:
String

Applies to:
*.APJ files

See Also:
TableInfo, TableType, TableNames, TableIndexes

VBA example:

Function GetNameLookup(Filename As String, TableType As String, Index As Integer) As String
    On Error Resume Next
    GetNameLookup = AriusProject.NameLookup(Filename, TableType, Index)
    If Err.Number <> 0 Then
        MsgBox AriusProject.ErrorMessage(Err.Number), vbCritical, "Error"
    End If
End Function

Usage:

Entering the following formula into a cell:

=GetNameLookup("C:\Users\Your_Username\Documents\Milliman\Arius\DemoFiles\Arius_Sample.APJ", "Input", 16)

returns “Paid Loss”.
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:**

```vba
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:


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:**

```vba
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.

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

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 AriusSystemnction
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

- “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:**

```vba
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
```