OLIVER WYMAN

Consulting Actuaries

Volume 3 | **Spring 2020**

PROFITING FROM PROPHET

MODEL HOUSEKEEPING

Editor's words: Welcome to the Spring 2020 edition of our Prophet modeling newsletter. This issue discusses the benefits and considerations of model consolidation and takes a closer look at user defined functions, an exciting new Prophet coding tool. You will also find helpful tips and tricks and a summary of the vendor-owned model showcasing US GAAP ("GAAP") long-duration targeted improvements ("LDTI") capabilities. Finally, all these topics should be viewed against the backdrop of the new Actuarial Standard of Practice ("ASOP") 56, Modeling, which takes effect in October 2020.

We hope you enjoy the newsletter. Please look for our next edition in Fall 2020!

IN THIS ISSUE

Executive Corner

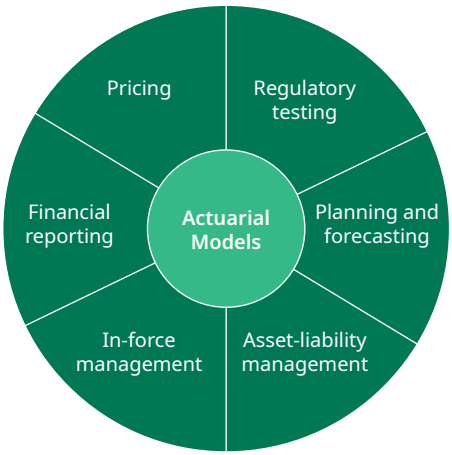
MODEL CONSOLIDATION: IS IT WORTH IT?

INTRODUCTION

Actuarial models serve as the backbone of an insurance organization's financial success and are heavily relied upon to understand expected future outcomes, satisfy regulatory requirements, and support strategic decisions. The new ASOP 56, Modeling, increases focus on the actuary's use of and reliance on models and ensuring they are robust and "fit for purpose".

Actuarial model development and maintenance often become decentralized as a natural consequence of the segregation of business units within an organization, even though there may exist a large overlap in modeling requirements. Exhibit 1 illustrates common actuarial business functions with a heavy reliance on actuarial models.

Exhibit 1. Common actuarial business functions



INDUSTRY ALERT!

ASOP 56: Modeling

ASOP 56[1] was developed recognizing the importance of modeling applications in actuarial science and underwent four exposure periods, with over 100 comment letters submitted. ASOP 56 serves to provide guidance to actuaries when performing actuarial services related to the design, development, modification, evaluation, selection, use or review of any type of model. ASOP 56 applies to the extent of the actuary's responsibilities, which may involve performing actuarial services related to an entire model or a subset of a model.

The Actuarial Standards Board ("ASB") voted in December 2019 to adopt ASOP 56 effective for work performed on or after October 1, 2020. As a next step, the ASB will review ASOP 38, Catastrophe Modeling, for any changes necessitated by ASOP 56.

1. ASOP 56 may be viewed here: http://lists.actuary.org/t/1284077/23691228/11011/2/

Under a decentralized modeling framework, individual business units are responsible for developing and maintaining their own models or groups of models. In contrast, under a centralized (or "consolidated") framework, a core modeling team is responsible for model development and maintenance, though business units often maintain formal ownership of their models, with access to modify them in "sandbox" environments for less strictly governed use cases, such as ad hoc analysis and sensitivities.

While many would agree that consolidating models is beneficial, companies are often hesitant to do so because of logistical complexity, resource requirements, or internal resistance. However, model consolidation promotes standardization across business units, improves efficiency and governance within the organization, and simplifies auditability and validation of the actuarial models.

The remainder of this article will further discuss benefits and challenges of both decentralized and centralized models, tips to consider when consolidating models, and how Prophet supports model consolidation.

DECENTRALIZED MODELS PRESENT BOTH BENEFITS AND CHALLENGES

Decentralized actuarial models certainly facilitate the flexibility required in today's actuarial modeling environment. However, decentralization may foster increased operational risk and longer-term inefficiencies. Key benefits and challenges of decentralized actuarial models are outlined in Exhibit 2.

Exhibit 2. Benefits and challenges of decentralized model development and ownership

Benefits	Challenges
Independence • Business units maintain autonomy around modeling decisions • Models have a clear owner within each business unit • Model issues and errors are isolated to the specific model	**Standardization** • Inputs and outputs may differ materially between models • Model output definitions may vary between models or systems, leading to possible misinterpretations of results • Modeling systems have different limitations
Flexibility • Each business unit can use the best-in-class system for the model purpose and business modeled • Model updates can be quickly implemented	**Efficiency** • Models may result in duplication of effort • Costs may be higher due to extra system licenses, multiple modeling environments, etc.
Customization • Business units can customize to the model purpose • Models only need to include necessary components	**Operational silos** • Communication between business units may be limited • Increased key person risk (e.g., only the dedicated model owner has knowledge of intricate model details)

MODEL CONSOLIDATION IS WORTH IT

Naturally, the primary advantages and disadvantages of model consolidation can be deduced by inverting Exhibit 2. However, a deeper dive further illustrates the tangible benefits that model consolidation can bring to insurance companies.

Model consolidation facilitates standardization across business units. It promotes consistency among modeling systems, inputs, modeling approaches, outputs, documentation, etc. Conversely, lack of standardization increases model risk. For example, an annual assumption update may become error-prone when multiple models require different data formats. Further, it is common for separate actuarial models of the same block of business (e.g., pricing, valuation, cash flow testing) to project diverging cash flows due to varying modeling approaches. Standardization improves the ability to compare and attribute results from different models.

Model consolidation creates long-term efficiencies by combining multiple teams. Reducing staff allocated to model development, decreasing turnaround time for analyses impacting multiple models, and increasing time available to validate and analyze results are concrete examples of common efficiencies achieved in a consolidated model framework.

Model consolidation simplifies auditability and external interactions. Consistency among models, coupled with a consolidated team that understands those models, results in efficient and effective conversations with external parties, such as auditors, regulators, and rating agencies, and also allows for a unified response to regulatory changes, such as LDTI and principle-based reserving (“PBR”).

TIPS FOR MODEL CONSOLIDATION

For many organizations, consolidation of actuarial models can seem a daunting task. Often, companies will choose to strategically consolidate specific aspects and functions of actuarial models, which allows them to retain a range of decentralization benefits while addressing several challenges with minimal effort, logistics, or resistance. There is no perfect approach to model consolidation, and a company should consider its own specific circumstances when designing a solution. However, the process can be made a little less daunting by following a few tips, as shown in Exhibit 3 on the next page.

Exhibit 3. Tips for consolidating models

Combine development of similar models	Combine separate model development teams into logical aggregated teams. For example, a company could combine development of permanent and term life models separately from the development of annuity models.
Merge models	Merge models where appropriate. For example, use a single valuation model for multiple reporting bases to reduce overhead (e.g., licensing costs, model refresh effort) and promote consistency. The increased efficiency and consistency typically outweighs the increased complexity.
Unify oversight	Establish a centralized team that oversees and ensures consistency between all actuarial models. This team should be responsible for ensuring consistency between inputs, calculations, coding standards, and outputs. Certain organizations have designated a global "model steward" oversight role.
Document, document, document	Prioritize adherence to strict documentation standards across all teams. Document key decisions, model comparisons, and attribution exercises. Comprehensive documentation will increase model transparency and facilitate the transfer of model duties.
Promote communication	Overcome operational silos by encouraging constant communication between teams, with a focus on modeling approaches and results. Communicating the benefits to each functional team and showing tangible results helps obtain buy in.

Consolidation of actuarial models

LEVERAGING PROPHET FOR YOUR MODEL CONSOLIDATION

Modeling software selection is often the most challenging decision facing an organization consolidating its actuarial models. The software of choice should not only be equipped to derive accurate results but also support the consolidation of end-to-end processes (e.g., assumption updates, downstream reporting processes, in-force management).

Prophet is well-suited for model consolidation when considering all components of the Prophet modeling suite.

Prophet promotes standardization through its standard libraries. Prophet's standard libraries include out-of-the-box functionality for integrated asset-liability modeling. Sharing standard libraries, such as the Asset Liability Strategy library, across various products and internal use cases promotes modeling consistency and generates efficiencies. When consolidating models, the open nature of libraries makes comparing and merging different versions of standard libraries relatively straightforward.

Prophet fosters robust governance and controls via its end-to-end modeling suite. The Prophet suite is ideally designed to support model consolidation and a single core modeling team through existing out-of-the-box functionality that provides segregated environments, formally defined roles and responsibilities, and a clear chain of "handoffs". For example:

- **Assumptions Manager** separates the assumptions stewardship process from model stewardship to ensure consistency of assumptions across multiple use cases via a controlled process
- **Prophet Enterprise** maintains segregated environments with pre-defined user access rights and provides end-to-end transparency by documenting the entire model run process
- **Prophet Control Centre** allows for the automation and control of the entire Prophet suite, minimizing manual processes, reducing risk of error, and ensuring compliance with internal governance and sign-off procedures

CONCLUSION

Actuarial modeling is a vital component of insurance company operations. Many organizations maintain a decentralized modeling structure, resulting in increased operational risk and other challenges.

Consolidating actuarial models is challenging as it comes with logistical complexity, resource requirements, and potential internal resistance. Prioritizing consolidation of certain aspects of actuarial business functions can result in concrete, short-term "wins", which often helps to overcome resistance to a broader consolidation effort.

Prophet is a strong platform to support model consolidation, due its open code, standard libraries, out-of-the-box audit tools, and transparency. This presents an opportunity for companies to realize synergies across business units as they incorporate Prophet's end-to-end solutions into their modeling ecosystem.

Prophet Model Health Check

With the effective date of ASOP 56 less than six months away, companies should be ensuring models are robust and fit-for-purpose. For a complimentary Prophet model “Health Check”, please reach out to the Oliver Wyman consultants named on the back cover.

Oliver Wyman Actuarial Modeling Software Survey

In addition, Oliver Wyman will be releasing an actuarial modeling software survey in Q4 2020. Please reach out with any topics or questions of interest. We value your insights.

What's New in Prophet

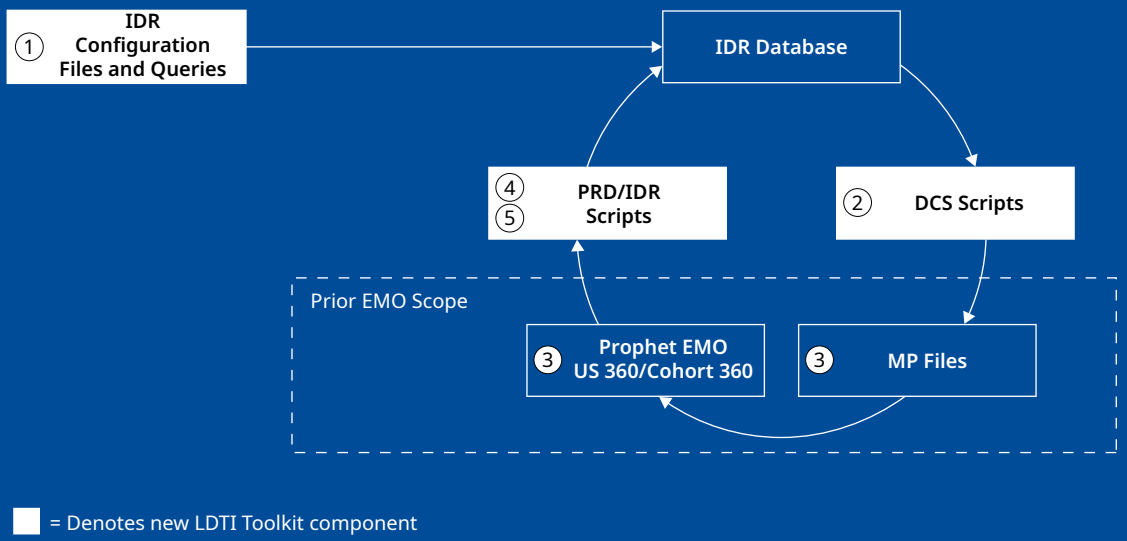
PROPHET LDTI TOOLKIT

LDTI fundamentally changes how insurers calculate and disclose US GAAP financials. LDTI introduces operational challenges in various aspects, particularly in data management. To illustrate the use of vendor software to address some of these challenges, FIS has released a sample data management suite for LDTI ("Prophet LDTI Toolkit") using the Insurance Data Repository ("IDR").

Prophet LDTI Toolkit provides a demonstration of FIS's end-to-end LDTI solution, leveraging new Prophet components to supplement the current US Life & Annuity 360 and US GAAP Cohort 360 example model office ("US 360 EMO"):

1. IDR configuration files and queries to establish a sample IDR database
2. Data Conversion System ("DCS") scripts to retrieve prior-period results from the IDR database and create model point files
3. Enhancements to the US 360 EMO model to run the model point files and calculate results for LDTI reporting
4. Prophet Results Database ("PRD") scripts to translate relevant US GAAP Cohort 360 results into SQL format
5. IDR scripts to transform and store the results in the IDR database for next-period reporting and other downstream use cases, such as ad-hoc analysis, results visualization, and management reports

As of the time of writing, this template is available for whole life products, with planned enhancements to product coverage, functionality, and visualization examples to be rolled out in stages over calendar year 2020.



In the Spotlight

DEFINING USER DEFINED FUNCTIONS

INTRODUCTION

User Defined Functions (“UDFs”), first introduced in Prophet 2018Q3, are modular blocks of code that can be repeatably called to execute calculations and return a value. A long-anticipated addition to the system, UDFs improve code maintenance without drastic changes to existing workspaces.

UDFs have tradeoffs that should be considered before implementation by a developer. The following exhibit discusses considerations in using UDFs.

Exhibit 1. Considerations in using UDFs

Benefits	Considerations
Cleaner variable code	Complexity introduced to the workspace via additional items to manage
Code modularity – shared code can be managed in one place	Newer functionality may create a learning curve for less experienced developers
Potential for increased model efficiency	Introduces model versioning limitations

USING UDFS

UDFs can be used to replace repetitive sections of code where a calculation uses multiple inputs and returns a single number within a variable. They provide a developer the ability to create and call a custom function in a similar fashion to system functions such as MAX or MIN. Primary uses of UDFs are variable organization, error reduction, simplified library maintenance, and code efficiency.

UDFs are stored at the library level and can be called from standard, extended, and t-dependent extended formulas. They are referenced with the following syntax in Exhibit 2:

Exhibit 2. Sample UDF snippet

```
Variable_Name := UDF.<FUNCTION_NAME> ( <Parameter_1> , <Parameter_2> , …
                , <Parameter_9> )
```

Exhibit 3 shows a code snippet of a variable calling UDF INTERP_LINEAR and assigning the return value to Array_Temp:

Exhibit 3. Example UDF

```
Array_Temp := UDF.INTERP_LINEAR (Index_Val_St, Index_Val_End _
            , Current_Read_T - Prev_T)
```

Editing UDF code is similar to editing extended formulas. For example, the UDF code editor shares familiar capabilities including code folding, syntax highlighting, and auto complete.

Model developers should be prepared to identify segments of code that are repetitive in nature, can be driven by input parameters, and return a single value or array. These code segments represent prime targets to be encapsulated within UDFs.

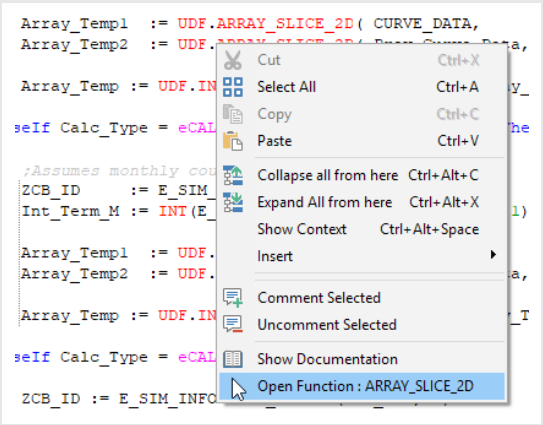
UDFs can also be a component of the standard libraries; however, at the time of writing, FIS has not introduced UDFs into the latest versions of most standard libraries. Exhibit 4 lists usage for the primary US libraries:

Exhibit 4. UDF usage by US libraries

Library	Current UDF usage
ALS Library (US) 2019Q4	32 unique UDFs called 154 times in 19 variables
US Life & Annuity 360 – Feb 2020	No UDFs
US GAAP Cohort 360 – Feb 2020	
US Life & Annuity – 2019Q3	
US GAAP Cohort – 2019Q3	

Quick UDF Tip

When viewing a variable in Formula Editor, you can right-click on a UDF, click “Open Function: [Function Name]”, and the UDF editor will open from the library

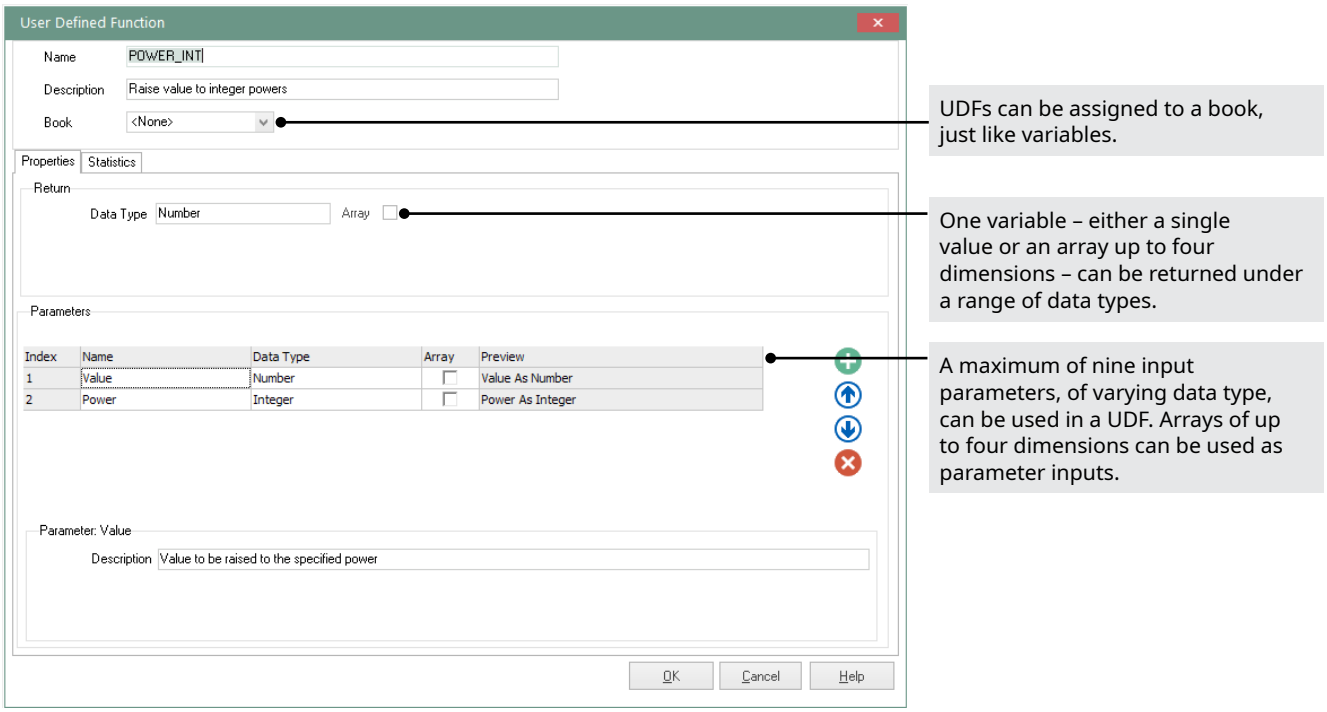


CREATING USER DEFINED FUNCTIONS

UDFs can be created in a library – just like a variable or indicator – either from scratch or by duplicating and modifying an existing function.

When creating a new UDF, the property box appears with a number of customization options. The POWER_INT UDF in the latest Asset Liability Strategy ("ALS") library is shown below along with comments.

Exhibit 5. UDF definition



After setting the UDF properties, the code editor can be opened and the function developed. The Prophet extended formula programming syntax is used, and a Return value is required for the function to be saved.

UDF EXAMPLE

This section takes a closer look at a UDF in practice. The POWER_INT UDF has been introduced to the ALS library for use in place of exponent-based calculations. Exponents, called through the "^" function, are classified in Prophet as a "complex" calculation, which generally impair performance. On the next page are examples of Prophet code before and after leveraging the UDF.

Exhibit 6. Code without UDF

```
Calc_Value  := (1 + Calc_Value) ^ (12 / Payment_Freq) - 1
```

Exhibit 7. Code with UDF

```
Calc_Value  := UDF.POWER_INT (1 + Calc_Value, INT(12 / Payment_Freq)) - 1
```

The following exhibit shows the code editor for POWER_INT and comments on the various sections of code. Notice the similarity between UDFs and standard extended formulas, the implication being that the learning curve for coding UDFs is small for experienced Prophet modelers.

Exhibit 8. Sample UDF

function POWER_INT(Value : Number; Power : Integer) : Number;

```
1  Local Value_Return As Number
2  Local Temp         As Number
3  Local bitvalue(0)  As Integer
4  Local counter      As Integer
5  Local quotient     As Integer
6  Local remainder    As Integer
7
8  Switch (Power)
9     ;Use a simple algorithm for commonly used powers 0,1,2,3,4,6,12
10    Case 0:  Value_Return := 1
11    Case 1:  Value_Return := Value
12    Case 2:  Value_Return := Value * Value
13    Case 3:  Value_Return := Value * Value * Value
14    Case 4:  Value_Return := Value * Value
15             Value_Return := Value_Return * Value_Return
16    Case 6:  Value_Return := Value * Value * Value
17             Value_Return := Value_Return * Value_Return
18    Case 12: Value_Return := Value * Value * Value
19             Value_Return := Value_Return * Value_Return
20             Value_Return := Value_Return * Value_Return
21
22    ;Otherwise use exponentiation by squaring
23    Case Else:
24       If Power > 0 Then
25          Temp := Value
26          quotient := Power
27       Else ;Power < 0
28          Temp := 1 / Value
29          quotient := -Power
30       EndIf
31
32       ;Convert Power to binary
33       counter := -1
34       Do
35          counter := counter + 1
36          remainder := MOD(quotient, 2)
37          quotient  := DIV(quotient, 2)
38          bitvalue(counter) := remainder
39       Loop While quotient > 0
40
41       Value_Return := 1
42       Do
43          Value_Return := Value_Return * Value_Return
44          If bitvalue(counter) = 1 Then
45             Value_Return := Value_Return * Temp
46          EndIf
47          counter := counter - 1
48       Loop While counter >=0
49 EndSwitch
50 Return Value_Return
```

Local variables act in place of private variables, and persist only for the duration of the function call.

For common exponents, the code multiplies the input value by itself instead. This code may appear ungainly, but is hidden from common view by being placed within a UDF.

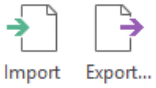
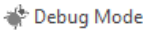
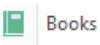
An alternative option is presented, but yields similar run-time performance as not using the UDF.

The Return keyword ends the UDF function and returns all values.

ADDITIONAL CONSIDERATIONS

UDFs are well supported by the Prophet Professional user interface, with functionality including but not limited to:

Exhibit 9. UDF user interface supported functionality

	• UDFs can be imported and exported from libraries and across workspaces
	• When used in variables, UDFs show up as a precedent similar to variables and tables
	• UDFs are included in library comparisons in a separate tab like variables and indicators
	• When stepping through code, you can step into and through UDFs • Breakpoints can be added within the UDF code
	• UDFs can be selected along with other functions in the Prophet function wizard
	• UDFs can be assigned to books • A variable in a locked book can call a UDF outside of the book

It is important to recognize UDFs for their intended use and inherent limitations. UDFs fit within the Prophet modeling worldview and may lack the range of capabilities found in pre-defined functions in other programming languages. They are intended to support and augment standard formula and extended formula variables, not replace them.

Exhibit 10. Limitations of UDFs

UDF limitations	Detail
System and library limitations	• Using standard libraries with UDFs will require a Prophet version upgrade, i.e., UDFs are only supported in Prophet 2018Q3 and later[1]
Input limitations	• A maximum of nine input parameters (which can be up to four-dimensional arrays) is available for managing inputs • No access to other variables within a product
Code limitations	• Unable to use standard read functions • Maximum of 250 lines, excluding blank lines and comments • No persistent internal variables • UDFs cannot call other UDFs
Output limitations	• Returns one value, which can be a single value or up to a four-dimensional array • PRINT is available, but not PRINT_TO_FILE

1. Note: Pre-UDF standard libraries may see limited ongoing vendor support

CASE STUDY

While the ALS library has been upgraded to make use of UDFs, a version without UDFs is available for reference in older US 360 model office packages (the latest package available at time of writing is October 2019) for US clients who have not upgraded past Prophet 9.0.4. This provided the opportunity to compare two out-of-the-box ALS products across two library versions:

Exhibit 11. UDF availability by workspace

Workspace	ALS library	UDFs
ALS Asset and Fund 2019 Q3	ALS Library (AF) – ALS 2019Q3	Yes
US 360 PBR EMO – Sep 2019	ALS Library (US) – 2019 Jul	No

The ALS product A_USMO from the US 360 PBR EMO workspace was first replicated in the ALS Asset and Fund workspace. This permitted a line by line comparison to be conducted between the native and replicated products using Prophet's comparison utility.

Below is an example of the APPLY_INT_TREATMENT UDF replacing a set of code that is otherwise duplicated in two separate ALS variables.

Exhibit 12. Code with UDF

```
Interest_Rate(Asset_Part) := UDF.APPLY_INT_TREATMENT( _
                               E_INFO.INT_TREAT_METHOD(Interest_Type_Used) _
                             , Interest_Rate(Asset_Part) _
                             , Part_Payment_Freq)
```

Exhibit 13. Code without UDF

```
If E_INFO.INT_TREAT_METHOD(Interest_Type_Used) = eTREATMENT_TYPE.Compound Then
    If payment_freq(Asset_Part) <> 1 Then
        Interest_Rate(Asset_Part) := ( 1 + Interest_Rate(Asset_Part) ) _
                                     ^ ( 1 / payment_freq(Asset_Part) ) - 1
    EndIf
ElseIf E_INFO.INT_TREAT_METHOD(Interest_Type_Used) = eTREATMENT_TYPE.Simple Then
    ;Convert simple to monthly rate
    Interest_Rate(Asset_Part) := Interest_Rate(Asset_Part) / 12
    If payment_freq(Asset_Part) <> 12 Then
       Interest_Rate(Asset_Part) := ( 1 + Interest_Rate(Asset_Part) ) _
                                    ^ ( 12 / payment_freq(Asset_Part) ) - 1
    EndIf
EndIf
```

The UDF code converts an interest rate based on the treatment method (simple/compound) and payment frequency. The input parameters are the treatment method, interest rate, and payment frequency.

This case study demonstrates the power of UDFs to streamline Prophet code and improve readability by shifting repetitive code away from key variables.

Tips & Tricks

INNER LOOP GROUPING

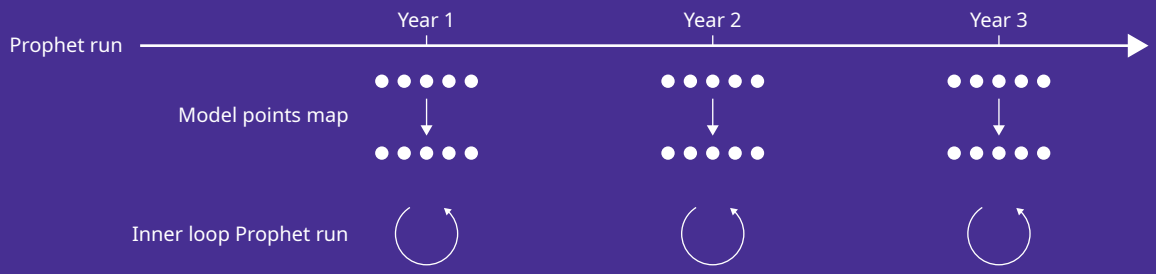
Prophet Nested Structures has introduced significant opportunities to extend model capabilities and support sophisticated calculations required under modern reporting frameworks. However, modeling nested runs can often be onerous in terms of runtime. Inner loop grouping within Nested Structure models can alleviate this by allowing model point compression when passing information to the inner loop.

Inner loop grouping is accomplished by calling a pre-compiled Data Conversion System ("DCS") script during the inner loop model point mapping process.

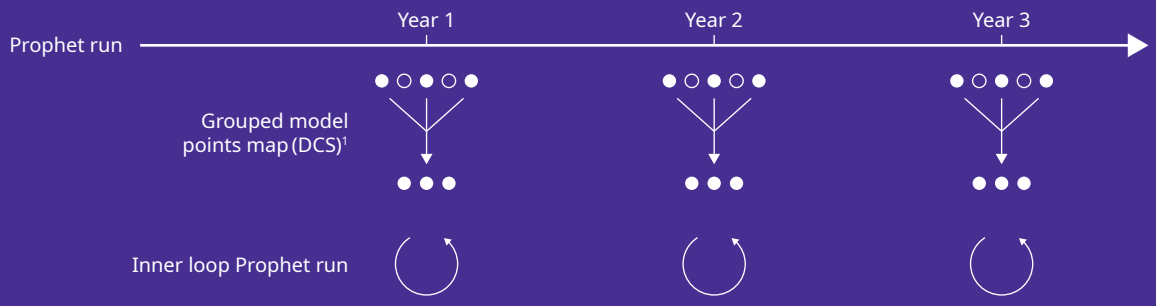
When defining inner loop groupings, the user should keep the following tips in mind:

- Grouping scripts can be created directly from Prophet Professional through an embedded version of DCS that automatically creates an input and output format that matches the model point map; however, less functionality is available than when leveraging DCS standalone
- Grouping scripts will behave the same in Prophet Enterprise and Push-to-PE as they do in Prophet Professional; DCS does not need to be installed separately on Prophet Enterprise worker machines
- Errors, warnings, and messages from the grouping script can be easily identified in the Prophet Professional runlogs as they are prefixed by "DCS>"

Without inner loop grouping



With inner loop grouping



1. A DCS program can be embedded within a run to perform the inner loop grouping

Tips & Tricks
ENUMERATIONS

Enumerations are very useful data types to map meaningful integer values used for modeling – such as calendar months, gender identifiers and premium modes – to strings. Through such a mapping, the modeler can benefit from both the modeling efficiency of integer-based switches and the descriptive nature of strings.

The use of enumerations has the potential to significantly enhance the readability of Prophet code, improve the maintenance of Prophet workspaces and reduce the risk of erroneous assumptions.

Model developers should keep these tips in mind when using enumerations:

- Instead of adding elements to the enumeration one-by-one, a list can be pasted from Excel into the Enumeration dialog box
- Enumerations can be used as constraints for variables and tables, similar to specified value constraints
- With the exception of extended formula internal variables, which can be defined as enumerations directly, variables must be modeled as data type 'Number' and generally rely on ENUM_TO_INT to leverage enumerations within definitions
- The Prophet Debugger will display both the enumerated value and name
- Be cautious when deleting an enumeration. Code that refers to missing enumerations must be reviewed manually, and references to missing enumerations within a product will cause run-time issues
- Enumerations can be compared and imported across workspaces
- As enumerations typically follow a distinctive naming structure (i.e. eEnumeration), Find/Replace functionality across products and workspaces works extremely well

For the ALS products compared, 14 ALS variables using 30 UDFs led to a few hundred redundant lines of code being removed.

CONCLUSION

UDFs are powerful tools that users can leverage to improve Prophet models. Many UDFs are used to reduce the code base but some have different goals. The UDF POWER_INT is intended to reduce the use of the time-intensive exponent calculation. Other UDFs, such as DAY_ADJ and CALENDAR_MTH_T, improve code structure and promote modularization and standardization.

The UDFs currently available out-of-the-box in the ALS library represent the tip of the iceberg with respect to UDF potential; however, as of the date of writing, FIS has left it to those licensing the system to expand the use of UDFs in non-ALS libraries. Prior to adoption, UDF capabilities and limitations should be reviewed in order to fully realize their value.

About Oliver Wyman

Oliver Wyman is a global leader in management consulting. With offices in 60 cities across 29 countries, Oliver Wyman combines deep industry knowledge with specialized expertise in strategy, operations, risk management, and organization transformation. The firm has more than 5,000 professionals around the world who work with clients to optimize their business, improve their operations and risk profile, and accelerate their organizational performance to seize the most attractive opportunities. Oliver Wyman is a wholly owned subsidiary of Marsh & McLennan Companies [NYSE: MMC], the world's leading professional services firm in the areas of risk, strategy and people. Marsh is a leader in insurance broking and risk management; Guy Carpenter is a leader in providing risk and reinsurance intermediary services; Mercer is a leader in health, wealth and career consulting; and Oliver Wyman is a leader in management consulting. With annual revenue of more than $14 billion and more than 65,000 colleagues worldwide, Marsh & McLennan Companies provides analysis, advice and transactional capabilities to clients in more than 130 countries.

The Actuarial Practice of Oliver Wyman has life, health, and property & casualty actuaries that advise financial institutions, insurance companies, regulators, and self-insured entities across a broad spectrum of risk management issues. With almost 375 professionals in over 20 offices across North America, the Caribbean and Europe, the firm's consulting actuaries provide independent, objective advice, combining a wide range of expertise with specialized knowledge of specific risks.

Fidelity National Information Services, Inc., shall have no liability in respect of the views and opinions expressed in this report.

For more information, please contact:

Dean Kerr
Partner
Dean.Kerr@oliverwyman.com
+1 416 868 7061

Justin Meade
Senior Consultant
Justin.Meade@oliverwyman.com
+1 816 556 4248

Guillaume Briere-Giroux
Partner
Guillaume.Briere-Giroux@oliverwyman.com
+1 860 723 5637

Matthew Zhang
Consultant
Matthew.Zhang@oliverwyman.com
+1 416 868 8712