Duck Creek
UNIVERSITY

ManuScript Standards and Best Practices

June 2022

Duck Creek
UNIVERSITY

The training materials and software are the confidential information and the proprietary property of Duck Creek Technologies and are provided subject to restrictions on use and disclosure as set forth in the applicable agreement. Due to continued product development, the information contained in these materials may change without notice. If you find any errors in the training materials, please report them to DCU.Learning@duckcreek.com. Duck Creek Technologies does not warrant that these training materials are error-free.

No part of these training materials may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, recording or otherwise or disclosed to any other person (including other employees of licensee) or third party—without the prior written permission of Duck Creek Technologies.

Duck Creek UNIVERSITY

Contents

Standards and Best Practices

This section is an abridged version of the ManuScript Standards and Best Practices article, which is located on the Solution Center.

In the absence of a client specific requirement indicating otherwise, fall back on using these Standards and Best Practices when building ManuScripts.

ManuScripts

These standards apply to ManuScripts as a whole or don't fit into any of the other sections.

Standard	Additional Information
When assigning the **Primary Value** in the ManuScript Properties, use the field that triggers Rating to occur. This will normally be **`data.SubmissionExecute`**, but may differ depending on the product.	The Primary Value in the ManuScript Properties is used when a value is requested from the ManuScript without specifying the specific field to calculate. If this situation comes up, it is most appropriate for the Rating process to kick off and return the overall premium for the product.
The root group for a product ManuScript should be *data*	A ManuScript must have a root group. By standard, this group should be named *data* for product ManuScripts.
Use a default of **0** for **additive factors**.	Factors being added together should default to **0** so that if the factor doesn't apply, it will not impact the overall calculation.
Use a default of **`1.0`** for multiplicative factors	Factors being multiplied together should default to **`1.0`** so that if the factor doesn't apply, it will not impact the overall calculation.
Use the fields named **`True`** and **`False`** instead of using string values of **`"True"`** and **`"False"`**	Always use the existing **`True`** and **`False`** fields when comparing Boolean values.
Never code a **URL**, **file path**, or **DSN** into a ManuScript field. Instead, create a new key in the **`Server.config`** file to hold this information.	Information that can vary by environment should be entered into an environment specific file such as the `Server.config` file in the Shared folder. Information in the `Server.config` can be accessed either using the Settings API object, or by using Runtime Replacement Tokens in requests.
Use field references rather than hard-coding values into fields	Code a value in as few places as possible. It is much better to code a value once, and reference it many times than it is to code it many times.
ManuScripts that contain ViewModel definitions should include **`VM`** in their file name.	To distinguish ManuScripts that use the ViewModel architecture from earlier patterns, insert VM to the ManuScript name, just prior to any version information. For example: `Carrier_ClientInformation_VM_1_0_0_0.xml`

Groups

These standards apply to groups.

Standard	Additional Information
A **public group** must contain three Private groups named **`Input`**, **`Output`**, and **`Private`**.	Append the word **Input**, **Output**, or **Private** to the name of the parent group when creating these private subgroups. The **Input** group is used to hold public fields to which a user or external system will supply values. The **Output** group is used to hold public fields that have system generated values such as Premiums. The **Private** group is used to hold all private fields. Example: A public group named **`MyGroup`** should have three private groups inside of it named: **`MyGroupInput`**, **`MyGroupOutput`**, and **`MyGroupPrivate`**.
A **coverage group** must have a path in the following format: `coverage[Type="Code"]`	The path must begin with **`coverage`**. To further identify the coverage a **`Type`** value is also specified. The Type value is normally a code uniquely identifying the type of coverage. Example: `coverage[Type="UIM"]`
A **coverage group** must contain an *Indicator* field in its Output subgroup.	Coverage groups should have a field named *Indicator*. The indicator field is used to indicate whether or not the coverage applies, regardless of what the Premium for the coverage may compute to. Downstream systems can check the Indicator value when trying to determine what is covered by the policy.
A **coverage group** must contain a **`Premium`** field in its *Output* subgroup.	Coverage groups must have a field named **`Premium`**. This is used calculate the Full Term Premium for the coverage.
A **coverage group** must contain a **`PremiumChange`** field in the *Output* subgroup.	Coverage groups must have a field named **`PremiumChange`**. This field is used to hold the value of the difference between the current transaction's **`PremiumWritten`** value and the prior transaction's **`PremiumWritten`**.
A **coverage group** must contain a **`PremiumWritten`** field in its *Output* subgroup.	Coverage groups must have a field named **`PremiumWritten`**. This field is used to calculate the term Written Premium for the coverage.
A **coverage group** should normally contain either a *Limit* or *Deductible* field in the Input subgroup.	Coverage groups should have either a *Limit* or *Deductible* field. These are either selected or entered by the user, or supplied by another process.
A **Line group** must have a path in the following format: `line[Type="Code"]`	The path must begin with *line*. To further identify the line a Type value is also specified. The Type value is the Line of Business code. Example: **`line[Type="GeneralLiability"]`**.
Group names should capitalize the first letter of each word.	Examples: AdditionalOtherInterest, RiskPrivate, ViolationInput

Fields

These standards apply to fields.

Standard	Additional Information
Numeric deductible fields should use **`-1`** to indicate that the coverage is not selected.	Numeric deductible fields may require options for both No Coverage and **0**. By using **`-1`** to indicate No Coverage, you can still use **0** as a valid option.
String deductible fields should use a value of **`"NoCoverage"`** to indicate that the coverage is not selected.	
Deductible fields must be **Public**.	
A **Factor field** should be **Private** unless there is a need for it to be changed to **Public**.	
A **Factor field** should have a data type of **Float**.	
An **Indicator field's** Path must match what is defined in the TransACTConfig file, which by default is **`Indicator.`**	This value is located in the TransACTConfig file at: TransActInfo/Premium/IndicatorPath
Indicator fields must be **Public**.	
Indicator fields must be Boolean.	
Coverage Indicator fields must be **True** if coverage applies, and **False** if coverage does not apply or is not selected.	If the indicator is True then the insured is covered for this regardless of what the Premium value for the coverage is.

Coverage Indicator fields should be **conditional** and the condition should be based on the coverage value supplied by the user (usually a *Limit* or *Deductible* value).	If the coverage has a Limit field then the *Indicator* should be True if the user has entered anything into the Limit field. If the user input field for the coverage is a string data type, then the Indicator should compare against the value that would indicate no coverage.
	Numeric Example:
	`BodilyInjuryInput.Limit > 0`
	String Example:
	`UninsuredMotoristInput.Limit <> "NoCoverage"`

Iterator fields should be created in the parent of the group being iterated through. Current Iteration iterator fields are an exception. **Current Iteration iterator fields** must live inside the group being iterated through.	Always place an iterator field in the parent group of the group being iterated through. This is both for performance and maintenance/debugging. However, iterators with a type of **Current Iteration** must be created inside the group they are iterating through in order to give the correct value, so these types are an exception.
Numeric Limit fields with defined options should use **`-1`** to indicate that the coverage is not selected or does not apply.	
String Limit fields with defined options should use **`"NoCoverage"`** to indicate that	

the coverage is not selected or does not apply.	
Limit fields must be *Public*	
Set MaxLengths on all Public string fields.	To avoid data truncation problems when converting policies between XML format and Database format using processes such as IDO or Shredding, always put a defined *MaxLength* on string fields. By defining a Max Length, the schema produced for the product will ensure that the new database column has enough space to fit all of the field's contents.
Do not use abbreviations for **field names** unless it is a well-known industry abbreviation.	Example: **`VIN`**
Field names should be descriptive of the data or function of the field.	Example: **`BaseRate`** would hold or calculate a base rate. **`VINIntegrationExecute`** would initiate an integration call to a VIN service.
Field names should capitalize the first letter of each word.	
Field names should avoid repeating the parent group's name.	Example: Instead of naming a field **`DwellingInput.DwellingLimit`**, the field should be named **`DwellingInput.Limit`**.
Yes/No fields should always list No first, then Yes.	
For **numeric fields with Defined Options**, the option values must be strictly numeric. Formatting such as commas and currency characters must be entered in the option captions.	
For **fields with Defined Options** and a Default value, the Default value must be one of the Defined Option values.	
For **fields with Defined Options**, the option values must match the data type of the overall field.	Example: if the field is Boolean it would be incorrect for a Defined Option value to be "True". "True" is a string value. Instead the Defined Option value must be **`1`**.
For **fields with Defined Options**, use Format Mask **`blankOption:(select)`** to supply a null value as the first option in a dropdown.	This is generally used to indicate to the user that they need to make a selection.
Coverage Premium fields must have a **Class** of **`Premium`**.	Full Term Premium fields for a coverage group must have a Class value of **`Premium`**
Premium fields must be Public.	
A Premium field in a **coverage group** must have a Path of **`Premium`**.	Full Term Premium fields for a Coverage group must have a Path value of **`Premium`**.

Coverage Premium fields must use a **Value** type of **Calculation**.	Coverage Premium fields should always be Calculations. Premiums that exist only to roll up child premiums can be Iterators, or straight field references.
Premium fields must have an *Ignore Calculation* expression that sets the Premium to **0** if the coverage's Indicator field is **False**.	For performance reasons every Premium calculation should have an *Ignore Calculation* expression that stops the calculation if there is no reason to calculate it.
PremiumWritten fields must be set to a value type of **`WrittenPrem.`**	
A **PremiumWritten field's** Path must match what is defined in the TransACTConfig file, which by default is **`written.`**	This value is in the TransACTConfig file at: TransActInfo/Premium/WrittenPrem
A **PremiumChange field's** Path must match what is defined in the TransACTConfig file, which by default is **`change`**.	This value is in the TransACTConfig file at: TransActInfo/Premium/PremiumChange
For **Request fields**, do not use **`//`** at the beginning of the *Response Path*.	Under certain circumstances this can result in incorrect results.
For **Select fields**, always supply an *"If no match"* value.	
For **Select fields**, the data type of the each Case value must match the data type of the overall field.	If the field Data Type is integer and you *Select on* a String field, each Case value must be an integer to match the Data Type of the overall field.
If you frequently use the same conditional Boolean phrase in several places, instead add a single conditional Boolean field, descriptively named, that checks the condition. That field can then be reused as a simple field reference to return the result of the condition.	This practice allows a single point of change if the parameter value of the condition needs to be modified instead of tracking down every field that was using that value in a condition. Example: **`VehicleInfo.TypeIsPrivatePassenger`** This technique should also be used for fields with defined options that will be checked frequently.
Avoid the use of **Always Execute**. It has a high performance impact.	

Tables

These standards apply to tables.

Standard	Additional Information
Factors, rates, or base premiums should be placed in a table named specific to their coverage, discount, or surcharge	Example: Create a table called **`LimitFactorsBodilyInjury`** that contains the factors for **BodilyInjury**.
Store all items which may change frequently (such as rates and factors) in tables rather than coded directly into fields	ManuScripts where frequently changed items are in tables are easier to maintain.
Tables with one Data Name should be named with the plural form of the Data Name	Example: a table with the single Data Name **`Factor`** might be named **`IncreasedLimitFactors`**
When possible create the table in the same format as it is shown in any provided documentation	When the ManuScript table is in the same layout as the requirements documents it enables the developer to cut and paste the data into the table, rather than manually typing in the values. It also makes the tables easier for business users to understand.
The Data Type for a Key must be compatible with the Key Values.	If the Key has Values of **`"One"`**, **`"Two"`**, and **`"Three"`** then the Key's Data Type must be *string*. If the Key has numeric Values of **`1`**, **`2`**, and **`3`** then the Key's Data Type must be *int*. If the Key has numeric values of **`1.0`**, **`2.0`**, and **`3.0`**, then the Key's Data Type must be **float**.

Pages

These standards apply to pages.

Standard	Additional Information
Avoid horizontal scrolling.	
Rating pages must be read-only.	
Rating pages should use *Execute ManuScript* to trigger the rating process when the page loads.	Execute ManuScript should be set to *Calculate and Accept* and be configured to calculate the field that triggers the rating process, unless that field is the same as the Manuscript's Primary Value, in which case it can be left blank.
Fields on Rating pages should not use *Compute Value*.	Any fields that are part of the rating algorithm will get values when the Execute ManuScript field is calculated, so there is no need for each individual field to be set to Compute Value.
Do not use **spacer fields** to add whitespace. Use CSS styling instead.	The Spacer field still exists in the base, but it should be used for assigning CSS styling, not as a way to manually add whitespace.

ViewModels

Standard	Additional Information
ManuScripts that contain ViewModel definitions should include **VM** in their file name.	To distinguish ManuScripts that use the ViewModel architecture from earlier patterns, insert **VM** to the ManuScript name, just prior to any version information. Example: `Carrier_ClientInformation_VM_1_0_0_0.xml`
List fields in the ViewModel in the order that they will appear on the page.	Duck Creek recommends that you add fields to the view model in the order that they appear on the page. This same order will appear in Page Builder, making it faster to add content to the view.
List fields in the ViewModel in the order that they will evaluate.	Duck Creek generally recommends that you add fields to the ViewModel in the order that they evaluate. In some cases, having ViewModel fields that are not ordered in this way can cause the ViewModel data to load incorrectly.
To limit the number of results that a grid or card can display on the page, the Maximum Iterations property must be set for the group to be associated with the control.	You can then subsequently set the Per Page property after adding the grid in Page Builder.
Action and Constant names cannot include any spaces.	The **Name** of an Action or Constant cannot contain spaces, however the **Value** of the Action or Constant can.
Rating ViewModels must be read-only.	
Rating ViewModels should use *Execute ManuScript* to trigger the rating process when the page loads	Execute ManuScript should be set to *Calculate and Accept* and be configured to calculate the field that triggers the rating process, unless that field is the same as the Manuscript's Primary Value, in which case it can be left blank.
Fields in Rating ViewModels should not use *Compute Value*.	Any fields that are part of the rating algorithm will get values when the Execute ManuScript field is calculated, so there is no need for each individual field to be set to Compute Value.
Use Constants for Heading captions and place the Constant in the same group as the fields it will accompany, unless that is an iterative group.	Duck Creek recommends that you add ViewModel constants to hold the labels for headers. If the constant defines a label used in an iterative group, you should add it within a ViewModel group that will be part of the XML output rather than the iterative group itself to avoid a potential error when the ViewModel containing the constant renders in Express.

Views

A View file name should conform to the following naming convention:

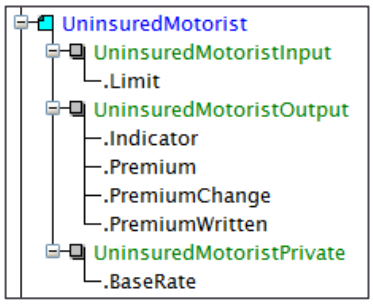
`[ManuScriptID]_[ModelCollectionName]_[ViewModelName]_View.xhtml`

Example:

`DCT_SampleProduct_3_0_0_0_MainInterview_ClaimsHistory_View.xhtml`

Standard	Additional Information
Avoid horizontal scrolling.	
Put each set of related fields in the same field container.	If you know that a set of fields will always appear together, but the position of that set of fields may change, it is recommended that you put each set of fields in the same field container. This makes it easier to move the set of fields around if you need to change their location in the view.
Add elements to the Stage design area by dragging from the View Model, rather than creating unbound elements.	Add groups, fields, and actions to the page from the View Model section of the page unless you specifically intend to use the data in a different way than how it would be added by default based on the data definition. This allows Page Builder to determine what is needed (certain containers, for example) and add it automatically, reducing the amount of manual work required from the configurator.
Use String Literal controls for messages and captions.	Use the String Literal control type for messages that show on the screen and manually added captions or labels, since in most cases these should not have labels and should be displayed as plain text.
Place tables in their own sections.	In most case, Duck Creek recommends that you place a table in its own section. This allows you to decide how to display the table when no data is present. You can display an empty table, a message saying no data is present, or both. To display a message, you need to set a show condition on the table that specifies to hide the table when no data is present.
Put carousel controls inside of a Panel when a show condition is needed.	To add a show condition to a carousel, Duck Creek recommends wrapping the carousel control in a panel container and putting the show condition on the panel.
Fields that you place in a Table Detail component must be present in the same view model that provides the data for the table row.	You cannot add an external view to a table detail row. If you want to link from within a table detail row to data defined in an external view model or ManuScript, Duck Creek recommends creating an action that loads a unique view of that data—for example, by opening the data in a modal window.
Iterative data must be displayed in an iterative control.	Tables, Cards, Carousels, and Tabs are the only elements that can display iterative content.
Limit content in a Carousel to no more than 30 iterations.	Duck Creek does not support more than 30 iterations in a Carousel.
Use the **`is-action`** modifier only once per page.	The Design System defines an element with this modifier as the one and only primary action for that page. Having multiple primary actions can diminish the visual importance that this modifier conveys.

Coverage Group Example



UninsuredMotorist Group

The main group for the coverage. This group must be public.

Scope	Public
Group Name	UninsuredMotorist
Coverage Type Code	UM
Group Path	coverage[Type="UM"]
Required	1
Maximum	1

UninsuredMotoristInput Group

All fields that the user can change are placed in this group. This group must be private, but the fields inside of it are all public.

Scope	Private
Group Name	UninsuredMotoristInput

UninsuredMotoristOutput Group

All public fields that have system generated values are placed in this group. This group must be private, but the fields inside of it are all public.

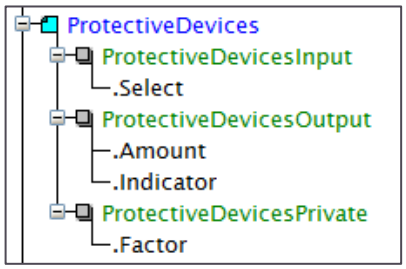
Scope	Private
Group Name	UninsuredMotoristOutput

UninsuredMotoristPrivate Group

All private fields are placed in this group. This group and all of the fields inside of it must be private.

Scope	Private
Group Name	UninsuredMotoristPrivate

Discounts and Surcharges Group Example



ProtectiveDevices Group

The main group for the coverage. This group must be public.

Scope	Public
Group Name	ProtectiveDevices
Coverage Type Code	ProtDev
Group Path	creditsAndSurcharges[Type="ProtDev"]
Required	1
Maximum	1

ProtectiveDevicesInput Group

All fields that the user can change are placed in this group. This group must be private, but the fields inside of it are all public.

Scope	Private
Group Name	ProtectiveDevicesInput

ProtectiveDevicesOutput Group

All public fields that have system generated values are placed in this group. This group must be private, but the fields inside of it are all public.

Scope	Private
Group Name	ProtectiveDevicesOutput

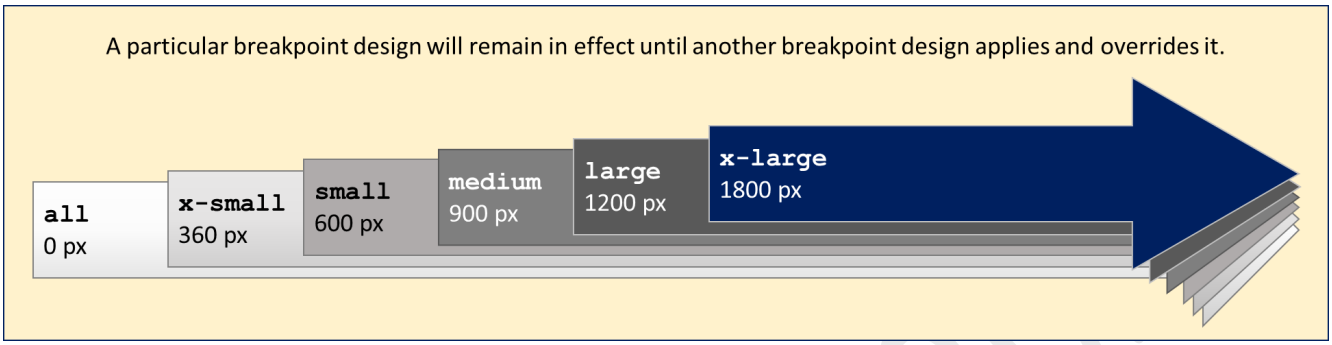
ProtectiveDevicesPrivate Group

All private fields are placed in this group. This group must be private, and all of the fields inside of it are also private.

Scope	Private
Group Name	ProtectiveDevicesPrivate

The Duck Creek Design System

Breakpoints



CSS Modifier Groups

Most CSS Modifiers are categorized into groups. Gaining an understanding of what modifiers in these groups do will increase your development productivity greatly.

Modifier	Additional Information
[breakpoint]-[1-12]	Sets the element to take up the specified number of columns within the layout at the specified breakpoint (and larger breakpoints unless overridden).
[breakpoint]-auto	Sets the element width to fill the remainder of the row. If multiple cells in the row are set to auto, they will each be of even width.
[breakpoint]-shrink	Sets the element to shrink to the width of its content.
[breakpoint]-show	Sets the element to become visible at the specified breakpoint (and larger unless overridden).
[breakpoint]-hide	Sets the element to become hidden at the specified breakpoint (and larger unless overridden).
default-[breakpoint]-[1-12]	Only available on containers. Sets the default number of columns that elements within the container should take up within the layout. Elements can override this setting by having their own breakpoint/size modifier.
default-[breakpoint]-auto	All elements in the container will be of even width.
default-[breakpoint]-shrink	All elements in the container will shrink to the width of their content.
order-[breakpoint]-[#]	Sets the sequential order of the element within the layout at the specified breakpoint (and larger unless overridden).
is-*	Specifies the state of the element.
has-*	Specifies a characteristic of the element.