Validation Rules Conventions

All Naming Conventions Are RFC 2119 and RFC 6919 compliant.

Metadata Conventions

1. The Validation Rule Name ***MUST*** try to explain in a concise manner what the validation rule prevents. Note that conciseness trumps clarity for this field.
2. All validation Rules `API` names ***MUST*** be written in PascalCase.
3. Validation Rules ***SHOULD NOT*** contain an underscore in the fields name, except where explicitely defined otherwise in these conventions.
4. A Validation Rule ***SHALL*** always start by `VR`, followed by a shorthand name of the object, followed by a number corresponding to the number of validation rules on the triggering Object, followed by an underscore.
5. The Validation Rule Error Message ***MUST*** contain an error code indicating the number of the Validation Rule, in the format `[VRXX]`, `XX` being the Validation Rule Number. \\1*
6. Validation Rules ***MUST*** have a description, where the description details the Business Use Case that is adressed by the VR. A Description ***SHALL NOT*** contain technical desccriptions of what triggers the VR - the Validation Rule itself ***SHOULD*** be written in such a manner as to be clearly readable.

Rule Conventions

1. All Validation Rules MUST contain a Bypass Rule check.

- The simplest bypass is a custom field of type `Checkbox`, set on the `User` Sobject, which you can Name `Bypass Validation Rules`. To avoid having to check how the consultant set the bypass, we recommend the API name always be `IsBypassVR__c`.
- If more granularity is needed, a consultant ***MAY*** want to create a `Hierarchical Custom Setting` to implement the bypass.
- This bypass can be added to the more common User-based bypass, but ***SHOULD NOT*** supplant it.

2. Wherever possible, a Consultant ***SHOULD*** use operators over functions.
3. All possible instances of `IF()` ***SHOULD*** be replaced by `CASE()`
4. Referencing other formula fields should be avoided at all cost.
5. In all instances, `ISBLANK()` should be used instead of `ISNULL`, as per this link.
6. Validation Rules ***MUST NOT*** be triggered in a casquading manner. \\2*

Examples

Name	Formula	Error Message	Description
VR01_OPP_CancelReason	`!$User.IsCanBypassVR__c && TEXT(Cancellationreason__c)="Other" && ISNULL(OtherCancellationReason__c)!$User.BypassVR__c && TEXT(Cancellationreason__c)="Other" && ISNULL(OtherCancellationReason__c)`	If you select "other" as a cancellation reason, you must fill out the details of that reason. [VR01]	Prevents selecting "other" less reason without putting a comment in. [VR01]
VR02_OPP_NoApprovalCantReserve	`!$User.IsCanBypassVR__c && !IsApproved__c && ( ISPICKVAL(Status__c,"Approved - CC ") \\|\\| ISPICKVAL(Status__c,"Approved - Client") \\|\\| ISPICKVAL(Status__c,"Paid") )`	The status cannot advance further if it is not approved. [VR02]	The status cannot advance further if it is not approved. [VR02]

> \1 While including an error code in a user displayed message may be seen as strange, this will allow any admin or consultant to find exactly which validation rule is causing problems when user need only communicate the end code for debugging purposes.* \2 Casquading Validation Rules are defined as VRs that trigger when another VR is triggered. Example: A field is mandatory if the status is Lost, but the field cannot contain less than 5 characters. Doing two validation rules which would trigger one another would result in a user first seeing that the fild is mandatory, then saving again, and being presented with the second error. In this case, the second error should be displayed as soon as the first criteria is met..*