Advanced Coding for Business Rules

When creating a business rule in Encompass, you can optionally apply conditions under which the rule is applied. If you need to define a condition for a scenario that is not addressed by the pre-defined condition types, you can use the Advanced Coding option to create your own custom expression. In addition to custom coding for conditions, you can also create custom code within the Milestone Completion, Field Data Entry, Field Triggers, and Loan Form Printing business rules.

Prerequisites
Knowledge of the following will help in creating advanced code:

- Visual Basic .NET
- Advanced Condition Editor
- Loan Custom Field Calculations

Visual Basic
The advanced coding business rule syntax is built on top of the Visual Basic .NET programming language. Before attempting to author an advanced coding rule, it helps to have a good understanding of the VB.NET syntax and logic flow, as those topics are not covered in this document.

Use of MsgBox or other UI-related functions is not supported when writing advanced code within Encompass business rules or triggers. These triggers and rules are enforced for all applications that may use the Encompass API, not just the Encompass client application. If a non-interactive application were to use the Encompass API with these triggers and rules in place, the application would hang indefinitely since there would be no way to respond. To implement interactive pop-ups and messages, it is recommended that you create those in a plug-in. While this may be a more complicated method, it is a far safer and more robust method.

NOTE: Familiarity with the VB.NET language is assumed throughout the rest of this document.

Advanced Condition Editor
The Advanced Conditions category of business rules provides the Advanced Condition Editor to create expressions to define custom conditions. This tool allows you to create a filter which writes the Visual Basic code for you.

Loan Custom Field Calculations
The advance coding business rule syntax also leverages the capabilities of the Encompass Custom Field Calculation engine which provides the ability to create calculations for both pre-defined and user-defined custom field IDs. In addition, the same calculations can be used in various places within the advanced code.

Help with Business Rules
Many additional resources are available to help you learn and use business rules.

Encompass Help
The online help system provides comprehensive information, including detailed descriptions of business rule features and tools and step-by-step instructions.

Documentation Library
The Documentation Library contains the latest versions of Encompass documentation, including:
- Installation and Administration guides, Chapter 12 Business Rules
- Loan Custom Field Calculations

Quick Reference Guide
Creating and Using Business Rules

Online Class
Setup and Configuration Part 3
Conditions

Business rules can include a condition under which the rule applies. For example, you can define a rule that applies only if the loan is for a purchase, or only if the subject property is in a particular state. Using conditions, you can define a wide array of rules that apply to all the different loan scenarios handled by your users.

Pre-Defined Conditions

To simplify the work of defining conditions, Encompass includes a number of pre-defined condition types to allow simple point-and-click selection. You can, for example, define a rule that applies only to loans originated in California by selecting the “Property State” condition type, and then selecting California from the list provided. The same is the case for conditions based on loan purpose, loan type, loan status, and so on.

Advanced Conditions

As you build your business rules, you may require a rule conditioned upon a field not provided in the pre-defined list, or that involves complex logic based on multiple field values. In these cases, use the Advanced Conditions option to author an expression that defines your custom condition.

When Advanced Conditions is selected, a text box appears where you can enter custom logic using Visual Basic .NET or you can enter a loan custom field calculation. The expression must evaluate to a Boolean (true/false) value. If the expression evaluates to true, the rule is applied; otherwise, the rule is ignored. If your expression results in an error of any kind, the rule will not be applied.
In the example below, if the purpose of the loan (field 19) equals Purchase and the numeric value of the purchase price (field 136) is greater than 999999, and the subject property state (field 14) is NY or NJ, then apply this rule.

**Advanced Condition Editor**

Instead of typing in the VB.NET code, you can use the Advanced Condition Editor to write the code for you.

1. Select **Advance Conditions** as the category type, and then click the **Edit Condition** icon on the right.

2. Click the **New** icon to add a new filter.

   **NOTE:** If you have existing filters, the New icon places the new filter at the bottom of the list. Select an existing filter and then click the **Insert** button to insert a new filter above the selected filter.

3. On the Add/Edit Search Filter window:
   - Type a Field to add as a filter.
   - Or, click the **Find** icon, select a field from the list, and click **OK**.
   - Or, click the **Find** icon, type one or more characters in the Find field and then click **Find**. The first occurrence of the characters in any column are found. Continue clicking **Find** to view additional occurrences. Select a field from the list, and click **OK**.

4. Select an Operator, enter (or select) a Value, and then click **OK**.

   The Operator and Value define how the field will be used. For example:
   Field ID = 1109 (Loan Amount), Operator = Greater than, Value = 250000

5. Click **OK**.

6. Repeat steps 2 through 5 to add more filters.

7. To use parentheses to control the order in which multiple filters are evaluated, click **Parentheses**, select the filters to group, click the **New** icon, and click **OK**.

8. Use the **AND/OR** button to determine how search filters are used together.
When finished, click **OK**.

In the example above, the expression evaluates to true if the loan is a refinance and the amount exceeds $250,000.

**Loan Custom Field Calculations**

A loan custom field calculation can be used in Advanced Conditions (refer to the "Loan Custom Field Calculations" document). When creating an expression for a business rule, it must evaluate to a Boolean (true/false) value.

The following expression is used to create a rule that is triggered when the loan amount (field 1109) exceeds $500,000.

\[\#1109 > 500000\]

The following expression enforces the business rule only if the loan was originated on or after Jan 1, 2010.

\[\text{[@MS.START]} > \#1/1/2010\]

The MS.START field represents the File Started date of the loan and the @ modifier forces Encompass to interpret the field value as a date. The comparison value, #1/1/2010#, is enclosed in # characters to represent a literal date value.

Conditions can also involve complex branching using the IIF statement. The following expression causes the rule to be enforced based on differing loan amounts for properties in California versus all other states.

\[\text{IIF([14] = "CA", [\#1109] > 200000, [\#1109] > 300000)}\]

All of the arithmetic and mathematical operations available to custom field calculations can be invoked as part of your condition.

\[\text{[\#110] + [\#111] + [\#112] >= 0.9 * [\#911]}\]

**Advanced Coding within Rules**

Custom code can be created within the Field Data Entry, Milestone Completion, Loan Form Printing, and Field Triggers business rules.

**Field Data Entry**

When creating a Field Data Entry business rule, you have the option of using the Advanced Coding option to create custom validation code for a specific field. In other words, the code will validate user input. This option is available only for a field rule defined with no other conditions. Field Data Entry rules are executed prior to the field's value being set.

The Advanced Coding option allows you to implement custom validation routines for individual loan fields that go far beyond what is permitted by the standard validation rules. You can use the values of other loan fields, employ arbitrarily-complex branching logic (if...then...else), or invoke a variety of built-in functions to determine whether a value being saved into the field satisfies your company's business rules. Additionally, you can customize the message Encompass displays to the user when invalid values are detected.

**To Create an Advanced Condition:**

1. On the menu bar, click **Encompass**, and then click **Settings**.
2. On the left panel, click **Business Rules**, and then click **Field Data Entry**.
3. On the Field Data Entry tool, click the **New** icon.
4. Type the name of the rule.
5. Select the channels where the rule will be applied.
6. Select **No** to always apply the rule.
7. Click **Add**.
8. On the Field Rule window, type the Field ID that the Value will apply to.
   - Or, click **Find** to select the field from an input form.
   - Select a form from the list on the left, right-click to select the field the rule will apply to, and then click **OK**.
9. Click the **Value Rule** tab, and click **Advanced Coding** for the rule type.
   - **NOTE:** You will see the Advanced Coding option only if the rule is defined as having no conditions (see step 6).
10. Enter code to define the valid field values.
Example
When authoring the code for an advanced business rule, you are essentially writing a VB.NET subroutine, in which you can use all of the language constructs of VB.NET to make the determination of whether the value input by the user is valid for the corresponding field. The following example will form the basis of the discussion of how to implement a custom business rule.

The example on the previous page reads as follows:
If the decimal value of the CX.ORIGLOANAMT field is greater than the loan amount, display a message to the user.

The example can be broken into the following components.

If...then...End If
The conditional syntax, If....then...End If, is taken directly from VB.NET and demonstrates the fact that VB.NET language elements can be employed without any special coding within your rule.

XDec() - If XDec(Value) > [#1109] then
The XDec() function is a function provided by the Encompass Custom Field Calculation engine and forces the conversion of the parameter to a decimal value. Refer to the "Loan Custom Field Calculations" document for additional information.

Value - If XDec(Value) > [#1109] then
The Value parameter is the sole argument passed into your business rule at runtime and represents the data input by the user into the specified field ID. In the example, the custom loan field CX.ORIGLOANAMT is the argument being passed into Value. Your code must determine whether the value stored in this parameter is appropriate for the field based on the current state of the loan. The Value is always passed as a string since one possible value for every field is the empty value (represented by the empty string). Therefore, it is unsafe to assume that Value will always have a numeric value even when your validation code is for a field such as the loan amount.

[#1109] - If XDec(Value) > [#1109] then
The square-bracket notation [#1109], is used to insert a reference to another field within the same loan. The # sign is a field modifier that stipulates that the field should be forced to be a number (in other words, it is the logical equivalent to XDec([1109])). The set of available field modifiers is defined in the document, "Loan Custom Field Calculations."

Fail() - Fail("This field's value cannot exceed the amount of the loan.")
The Fail() subroutine is the function your custom rule should invoke if you have determined that the Value is inappropriate for the current field. The Fail() routine takes one parameter: the message to display to the user. All processing within your business rule stops at the point where you call Fail(). No code is executed beyond that line.

If a validation rule reaches its end without calling Fail(), the Value is assumed to be valid for the field. If any exception occurs while executing the business rule, the validation is considered to have failed and the user will receive a notice indicating an error has occurred in the rule.
Additional Examples

Within your rule definition, you may declare and use variables, author loops using for, for each or while, or invoke any standard functionality provided by the .NET Framework within the System namespace. You may not invoke any function that results in the display of a user interface, nor should you spawn additional threads to perform asynchronous processing. Because your validation rules are enforced at the business level of the Encompass application, these rules may be invoked in a scenario when there is no user interface (such as when using a custom application written using the Encompass API).

The following example could be used to enforce restrictions on the loan amount based on the selected loan program. The Select Case syntax is used for branching instead of If...then...else, but that choice is entirely a cosmetic one.

```vbnet
Select Case [1401]
    Case "5/1 ARM"
        If XDec(Value) > 650000 then
            Fail("The loan amount cannot exceed $650,000 for a 5/1 ARM.")
        End If

    Case "3/1 ARM"
        If Value <> "" And XDec(Value) < 30000 then
            Fail("The loan amount must be at least $30,000 for a 3/1 ARM.")
        End If

    Case "30 Year Fixed"
        If XDec(Value) > 450000 then
            Fail("The loan amount cannot exceed $450,000 for a 30 year Fixed.")
        End If
End Select
```

The next example compares the value being set into the field to the sum of the several fees pulled from the GFE.

```vbnet
Dim sum as Decimal = 0
Dim i as Integer
Dim fieldIds as String() = new String() {"454", "1093", "640", "641", "329", "L228", "L230"}
For i = 0 to fieldIds.Length - 1
    sum = sum + XDec(Fields(fieldIds(i)))
Next
If XDec(Value) > sum then
    Fail("The discount value cannot exceed the sum of the prepaid charges.")
End If
```

The above example demonstrates the use of the Fields collection, which provides access to the set of loan fields. This collection can be used whenever the ID of the field you are retrieving is determined at runtime. The square-bracket syntax [FieldID] is equivalent to the expression Fields("FieldID").
Milestone Completion

Use the Advanced Conditions tab of the Milestone Completion rule to specify additional conditions that must be met in order to complete the milestone. For example, you can create custom code to require the completion of three additional fields when a loan exceeds a specified amount in a particular state or county.

1. On the menu bar, click Encompass, and then click Settings.
2. On the left panel, click Business Rules, and then click Milestone Completion.
3. On the Milestone Completion tool, click the New icon.
4. Type the name of the rule.
5. Select the channels where the rule will be applied.
6. Select No channel selected to apply the rule to a loan where none of the channel options are selected.
7. Select No to always apply the rule.
8. Or, select Yes to create a condition when the rule is applied.
   • Select a category for the condition, such as Loan Purpose.
   • Select a value for the category, such as Purchase.
   • If you select Current Role for the category, you can configure the rule to only be applied when a specific role is assigned to finish a specific milestone. After selecting Current Role, click the Lookup icon (magnifying glass), and then select the specific role. Then select the specific milestone from the drop-down list.
9. Select Advanced Conditions as the category condition, and then click the Find icon on the right.
10. Click the New icon to add a new filter.

NOTE: If you have existing filters, the New icon places the new filter at the bottom of the list. Select an existing filter and then click the Insert button to insert a new filter above the selected filter.
11. Type the code in the Advanced Coding window, and then click OK.
12. On the Add/Edit Search Filter window:
   • Type a Field to add as a filter.
   • Or, click the Find icon, select a field from the list, and click OK.
   • Or, click the Find icon, type one or more characters in the Find field and then click Find. The first occurrence of the characters in any column are found. Continue clicking Find to view additional occurrences. Select a field from the list, and click OK.
13. Select an Operator, enter (or select) a Value, and then click OK.
   • The Operator and Value define how the field will be used. For example: Field ID= 1109 (Loan Amount), Operator = Greater than, Value = 250000.
14. Click OK.
15. Repeat steps 2 through 5 to add more filters.
16. To use parentheses to control the order in which multiple filters are evaluated, click Parentheses, select the filters to group, click the New icon, and click OK.
17. Use the AND/OR button to determine how search filters are used together.
When finished, click **OK**.

**Example**

The example below tests if field FR0112 (number of years at present address) is less than 2. If that is true, display an error message.

```
When [#FR0112] < 2.00

Fail("At least 2 years of residence is required to complete the milestone.")

End if
```

**Advanced Coding**
Loan Form Printing

Use the Advanced Coding tab of the Loan Form Printing rule to create your own advanced coding rules that must be satisfied before a form can be printed.

1. On the menu bar, click Encompass, and then click Settings.
2. On the left panel, click Business Rules, and then click Loan Form Printing.
3. On the Loan Form Printing tool, click the New icon.
4. Type the name of the rule.
5. Select the channels where the rule will be applied.
6. Select No to always apply the rule.
7. Or, select Yes to create a condition for when the rule is applied.
   • Select a category for the condition, such as Loan Purpose.
   • Select a value for the category, such as Purchase.
8. Click Add.
9. On the Print Form Suppression Rule window, click Find.
10. Click the Standard Forms or Custom Forms tab, select the folder containing the form for which you want to create a rule, select the form, and click Select.
11. Click the Advanced Coding tab to write your own code to define the print suppression rule.
12. Select the check box to allow blank forms to be printed even if the required fields have not been completed.
13. When finished, click OK.
Triggers

Triggers provide the unique ability to modify the contents of a loan based on user action or data changes within the loan file. Although Encompass provides an interface for quickly defining simple triggers that involve very simple logic, your business requirements may demand much more complexity than is provided by default.

**NOTE:** For Trigger rules, if you select *Update the value of one or more fields* as the action to take when an event is triggered, you can set the specified field value using a custom calculation. For detailed information, refer to the "Loan Custom Field Calculations" document.

Just as with the Field Data Entry and Milestone Completion rules described earlier, the Triggers interface in Encompass allows you to author code in VB.NET to represent your custom logic. Your code is executed whenever the conditions of the trigger are met, such as when a field is set to a particular value.

*(NOTE: If you are creating a field trigger that runs advanced code to update fields, consider Dynamic Data Management (DDM). DDM provides a Data Table option that eliminates advanced code and makes it easier to create and maintain rules for updating fees and fields. For more information, see Dynamic Data Management User Guide.)*

Within your advanced code, you can:

• Read or write the values of the loan's fields
• Determine the prior and new values of the field the user modified
• Perform branching and conditional logic
• Use any of the functions natively supported by the VB.NET language

The advanced code for a trigger should be written as a VB.NET subroutine which takes three parameters: FieldID, PriorValue, and NewValue. These variables represent the ID of the field that was changed as well as its prior and new values, respectively. Unlike advanced code written for Field Data Entry rules, which is executed prior to the field's value being set, a trigger is activated after the field's value has changed.

Because a trigger can modify other field values in the loan, any rules or triggers associated with those fields will also be fired. However, a single trigger will never run recursively, meaning that if a trigger makes a change that causes itself to be fired a second time (or it causes a second trigger to run that would in turn fire the first trigger again), the trigger will not be executed. This prevents the possibility of an infinite loop caused by triggers firing one another in succession.

1. On the menu bar, click *Encompass*, and then click *Settings*.
2. On the left panel, click *Business Rules*, and then click *Field Triggers*.
3. On the Field Triggers tool, click the *New* icon.
4. Type the name of the rule.
5. Select the channels where the rule will be applied.
6. Select *No* to always apply the rule.
7. Or, select *Yes* to create a condition for when the rule is applied.
   • Select a category for the condition, such as Loan Purpose.
   • Select a value for the category, such as Purchase.
8. Click *Add*.
9. In the Activation section of the window, select the *Activation Type*.
   • If you select *Field value modified*, or *Milestone completed*, you will be asked to enter additional information as described below.
10. In the Action section of the window, select *Run advanced code*.
11. In the Advanced Code area, enter the code that will that will be run when the action is triggered.
12. Click *OK*.
13. When finished adding event details, Click *OK*.
14. Add additional field events as required, and then click *Save*. 

---

10 Advanced Coding for Business Rules

October, 2018
Examples

The examples illustrate a few of the commonly used techniques that you can take advantage of when writing your custom trigger code.

The following example uses the Subject Property State field (14) to determine the Broker Company Name (315) that is presented on the 1003 Page 3 form. The value of field 315 is set to one of three possible values based on the data input into field 14. To set a field value, use the same bracket ([ ]) notation as when retrieving a field value.

![Example Action Table]

The next example sets the value of a set of custom fields based on the loan amount (field 1109) and interest rate (field 3).

```
IgnoreValidationErrors

[CX.MYPROFIT] = [#1109] * [#3] * 0.02 / 100.0

If [19] = "NoCashOutRefi" then
    [CX.MYPROFIT] = 2 * [#CX.MYPROFIT]
End If
```

The first line of the trigger is the keyword IgnoreValidationErrors, which can be used only in triggers. This statement tells Encompass to ignore any field rules that may prevent changes to the fields modified during the execution of the trigger. For example, if you had a field rule for the CX.MYPROFIT field that restricted access to the field for certain users, then your trigger would fail when executed by those users because the rules would be enforced. By using IgnoreValidationErrors, you enable the trigger to work regardless of the field rules currently in place.

Accessing and Updating Non-Field Data

During the execution of your trigger, you may need to check or update the status of a milestone, task or document within the loan file. For example, when a certain field is populated, you may want to mark a Milestone as being completed. Because there is no fixed set of Field IDs for this type of dynamic data, alternative methods are provided for reading and writing this data.

Read-only access to most non-field can be obtained through the use of virtual field IDs. A virtual field ID behaves like a normal field ID but is dynamic based on the object it is referencing. For example, to retrieve the date the “Credit Report” document was received, you can use the virtual field ID [Document.DateReceived.Credit Report]. Note that the document’s title appears as part of the field ID, indicating that the field ID is dynamic based on the document you are referencing.

The following code uses the date the Credit Report was received to set the value of a custom field.

```
if [@Document.DateReceived.Credit Report] > #3/15/2010# then
    [CX.REPORTDATE] = [Document.DateReceived.Credit Report]
End If
```

Virtual fields are limited in their use because they are read-only. The following code is invalid:

```
[Document.DateReceived.Credit Report] = Today
```

To make changes to data represented by virtual fields, you will need to use one of the special functions defined on the next page, based on the type of object you want to manipulate.
## Milestone-Related Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milestones.IsComplete(x)</td>
<td>Returns a Boolean indicating if the specified milestone is complete. The value x should represent the name of the desired milestone.</td>
<td>Milestones.IsComplete(&quot;Qualification&quot;)</td>
</tr>
<tr>
<td>Milestones.SetComplete(x, optional isComplete)</td>
<td>Marks a milestone as completed or marks is as incomplete of the optional isComplete flag is passed and set to false.</td>
<td>Milestones.SetComplete(&quot;Qualification&quot;)</td>
</tr>
</tbody>
</table>

## Task-related Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks.Exists(x)</td>
<td>Determines if the specific task is present in the loan. The value x should represent the title of the task.</td>
<td>Tasks.Exists(&quot;Contact Borrower&quot;)</td>
</tr>
<tr>
<td>Tasks.IsComplete(x)</td>
<td>Determines if the specified task has been completed. If the task is not present, this method returns false.</td>
<td>Tasks.IsComplete(&quot;Contact Borrower&quot;)</td>
</tr>
<tr>
<td>Tasks.SetComplete(x, optional isComplete)</td>
<td>Marks a task as completed or as incomplete if the isComplete parameter is passed as false.</td>
<td>Tasks.SetComplete(&quot;Contact Borrower&quot;, true)</td>
</tr>
</tbody>
</table>

## Document-related Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents.Exists(x)</td>
<td>Determines if a document with the specified title is present in the loan.</td>
<td>Documents.Exists(&quot;Credit Report&quot;)</td>
</tr>
<tr>
<td>Documents.IsReceived(x)</td>
<td>Determines if a document with the specified title has been received.</td>
<td>Documents.IsReceived(&quot;Credit Report&quot;)</td>
</tr>
<tr>
<td>Documents.IsExpired(x)</td>
<td>Determines if the specified document has expired.</td>
<td>Documents.IsExpired(&quot;Credit Report&quot;)</td>
</tr>
<tr>
<td>Documents.IsOrdered(x)</td>
<td>Determines if the specified document has been ordered.</td>
<td>Documents.IsOrdered(&quot;Credit Report&quot;)</td>
</tr>
</tbody>
</table>

The following example demonstrates how to mark a task as completed based on whether a document has been received.

```plaintext
if Documents.IsReceived("Credit Report") then
    Tasks.SetComplete("Pull borrower credit")
End If
```