

# Lesson 6 Validation

As a configuration developer, you want to be able to verify the data entered by the user. If the data is incorrect (wrong format, missing information, inconsistencies between fields) then you want to prevent the save. You also want to be able to highlight widgets and show warning/error messages as needed. In this lab, you will make several configuration changes to improve the validation in TrainingApp.

## 6.1 Prerequisites

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.

<http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.



### Activity

## 6.2 Lab: Field-level validation in the UI

*"Every Auto Repair Shop has a license field that can be specified on the Company Info card. Currently, TrainingApp doesn't validate this field and allows saving license numbers in any format. Configure TrainingApp to meet the following requirements:*

- Auto Repair Shop licenses must be in the following format: the first character must be a lower-case "a" character, followed by any lower-case alphabet character, then 5 digits. Example: aw-36292
- TrainingApp should display a watermark and tooltip to guide the user
- License numbers cannot contain the string '777'. If the license number contains '777' then the save shouldn't be allowed, and the user should be notified that the license number is invalid.
  - Insurance company business analysts

### 6.2.1 Investigation

#### 1. View the current widget behavior

- a) Log in as Alice Applegate

- b) Search for Burlingame Saab
- c) In the sidebar menu, select Details
- d) Click Edit
- e) Enter any alphanumeric value in the License field
- f) Click Update

## 2. Open the PCF file in Studio

- a) To open the PCF, use ALT + SHIFT + E

### 6.2.2 Configuration

In this part of the lab, you will configure the License Text Input field to show the user a watermark.

#### 1. Modify the License widget to show a watermark

- a) Modify ABContactDetailsCompanyDV.pcf so that the ABAutoRepairShopLicense (ID: ABAutoRepairShopLicense) field shows the following watermark:

a#-#####



In this part of the lab, you will configure the AutoRepairLicense field to match the pattern of the field watermark.

#### 2. Modify the License widget to validate the input so that it matches the watermark

- a) Modify ABContactDetailsCompanyDV.pcf so that the ABAutoRepairShopLicense field matches the pattern of the watermark:
  - The first character must be a lower case "a" alphabet character
  - The second character must be any lower case alphabet character
  - The third character is a dash
  - The fourth, through eighth characters must be numbers

In this part of the lab, you will configure the AutoRepairLicense field to validate the value before it is committed. If the license number contains the string "777", then the user should be notified that license number is invalid.

### 3. Modify the License widget to validate the input based on business logic

- a) If the license number contains the string “777”, then the user should be notified that license number is invalid

#### 6.2.3 Verification

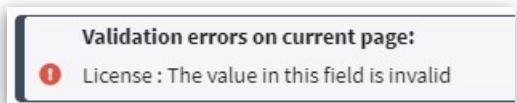


##### Activity

Verify the work you have done

###### 1. Verify the watermark and pattern

- a) In TrainingApp, deploy your changes to the PCF file
- b) Edit the Details for Burlingame Saab
- c) For the License field, enter **ab-12345** and verify that there is no field format warning
- d) Click Update
- e) Click Edit
- f) Enter **a1-12345** and click Update
- g) Verify that there is a validate error message:



###### 2. Verify the validation expression

- a) Change the license field value to ab-12777
- b) Click Update
- c) Verify the info bar error message:



- d) Enter ab-12345 as a valid license field value
- e) Click Update
- f) Verify that you were able to commit the changes



## Hint

An overview of the syntax for validator patterns is shown in the following table. **Note:** this is only a high-level overview. For a complete listing of syntax for validator patterns, consult the Configuration Guide for your product.

.	Any character
?	Indicates zero or one occurrence of the preceding element
+	Indicates one or more occurrences of the preceding element
*	Indicates zero or more occurrences of the preceding element
{X}	Preceding item is matched exactly X times
{X, Y}	Preceding item is matched at least X times, but not more than Y times. (technically X is minimum, Y is maximum)
.+	Any non-empty string (at least one or more characters)
\	Escape character
[ ] ( ) { } . * + ?	Characters that are not treated as literals and must be escaped

[charRange] {X}	Characters in character range, which can be any combination of 0-9, a-z, and A-Z. X means the exact number of characters we need from the charRange
[charRange] {X, Y}	From X to Y characters in charRange
@ - \$	Most other characters are treated as literals such as a dash (-)
\.	A period, where the backslash is the escape character

#### Examples:

.+@.+	Validates an email address. All of these are valid values: <a href="mailto:a@b.com">a@b.com</a> <a href="mailto:william@andy.com">william@andy.com</a> <a href="mailto:chris@SUCCEED.com">chris@SUCCEED.com</a>
[0-9a-zA-Z] {3}-[0-9] {3}	This validates the following format: 3 alphanumeric characters, a dash and 3 digits. All of these are valid values: 999-999, AB9-333, A7C-444
[0-9] {3}-[0-9] {2}-[0-9] {4}	This validates the following format: 3 digits, dash, 2 digits, dash and 4 digits. E.g.: 444-55-6666

## 6.3 Lab: Field-level validation in the Data Model

*"The insurance company requires that all bank account numbers follow the pattern XX-XXXX, where X is a digit from 0 through 9. The compliance with the pattern must be enforced in the UI as well as through APIs. Without needing to modify existing widget properties in the application, we also require that there is a watermark identifying the pattern. The validation error message for an invalid bank account value must be localizable." – Insurance company business analysts*

### 6.3.1 Configuration

#### 1. Create a validator display key error message

- a) Create a display key in display.properties that reads:

"A bank account number must be two numbers, a hyphen, and then four numbers. For example, 12-3456."

## 2. Create a field validator

- a) Create new ValidatorDef in fieldvalidators.xml
- b) Specify the attributes to meet the lab requirements

## 3. Associate the field validator with an entity element

- a) Add an entity field validator for the AccountNumber field in the BankAccount entity

## 4. Make the necessary changes to the AccountNumber Text Cell

- a) Make the necessary changes to the AccountNumber Text Cell in BankAccountsLV.pcf to display the correct watermark.

## 5. Deploy your changes

- a) In Guidewire Studio, restart (stop then debug) the server



### Hint

For an overview of the syntax for validator patterns, see the Hint located in the *Configure the pattern* section previously in this module on Validation. For a complete listing of syntax for validator patterns, consult the Configuration Guide for your product.

## 6.3.2 Verification



### Activity

Verify the work you have done.

#### 1. Verify the entity field validator behavior

- a) In TrainingApp, add a new bank account for the Burlingame Saab contact
- b) Verify the ##### watermark displays for the Account Number Text Cell
- c) For the account number, try to enter A1-B234
- d) Verify that validator error shows with the message you created
- e) Enter 12-3456 as the account number
- f) Click Update
- g) Confirm that a new bank account was added for Burlingame Saab without an error message

## 6.4 Lab: Validation rules

*"We have to make sure that the 'Married filing jointly' and 'Married filing separately' tax filing statuses are only allowed when the Marital Status of the person is Married. We want the following behavior to be enforced: if the Tax Filing Status is set to Married filing jointly/Married filing separately and the Marital Status is NOT married then the save must be rejected with an error message and the Tax Filing Status field must be highlighted. The error message must be the following: 'Married filing jointly/Married filing separately is only allowed when contact is married.'. The error message must be localizable." – Insurance company business analysts*

### 6.4.1 Configuration

#### 1. Create a new Validation Rule

- a) To implement this user story, use the existing ABContactValidationRules Rule Set
- b) Reject the save using the `loadsave` validation level

#### 2. Deploy your changes

- a) To deploy the new rule, select Run menu → Reload changed classes in Studio
- b) To deploy the new display key, use ALT + SHIFT + L in the browser



### Review

Working with Gosu Rules

If needed, review how to work with Gosu Rules, Rule Set Categories, and Rule Sets. Refer to the Gosu Rules lesson in the InsuranceSuite Fundamentals – Kickstart course.

## 6.4.2 Verification



### Activity

Verify the work you have done

#### 1. Verify the entity field validator behavior

- a) In TrainingApp, search for Eric Andy
- b) Go to the Details page
- c) Click Edit
- d) Set the Tax Filing Status to Married filing jointly
- e) Set the Marital Status to Single
- f) Click Update
- g) Verify that you see the error message and the field is highlighted
- h) Set the Marital Status to Married
- i) Click Update
- j) Verify that the data was successfully saved



## 6.5 Lab Solution: Field-level validation in the UI

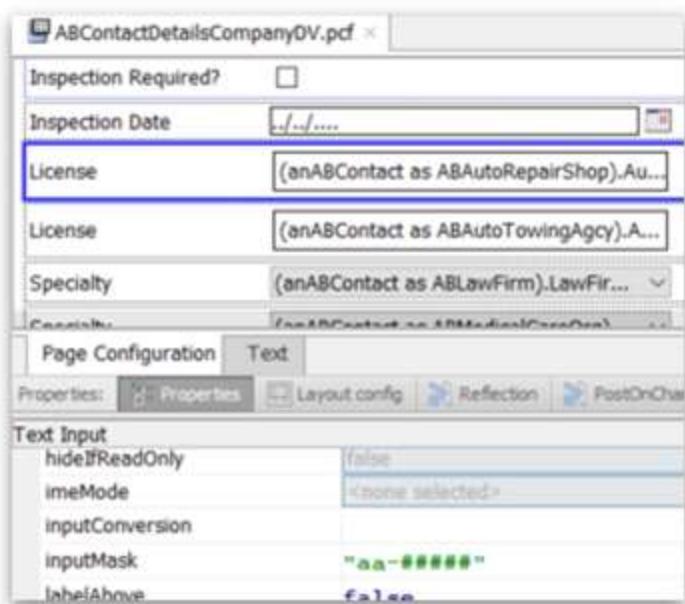


### Solution

Exact details on how to complete the lab.

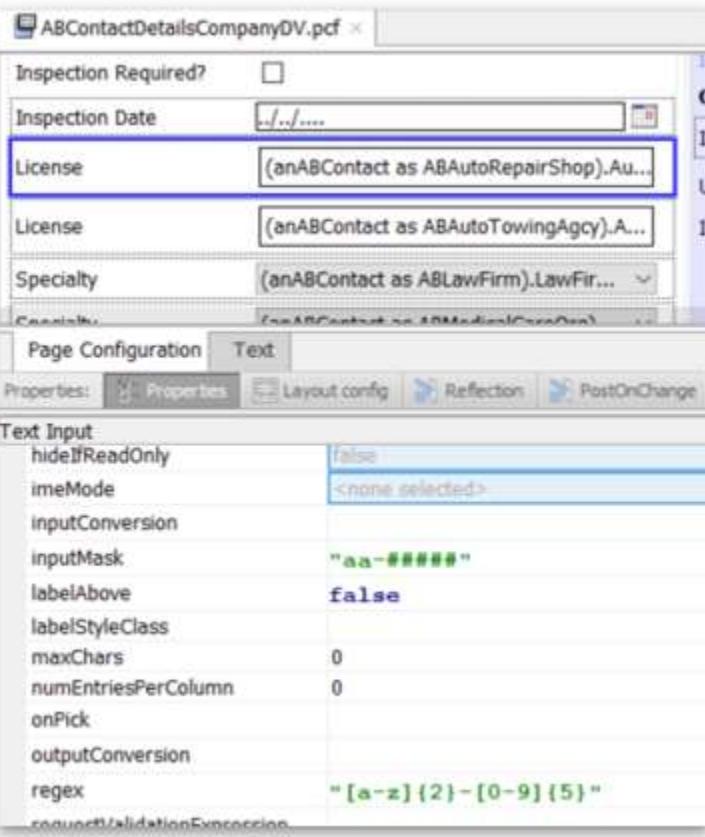
#### 1. Configure the inputMask property of the widget.

- Type "a#-#####" in the inputMask field.



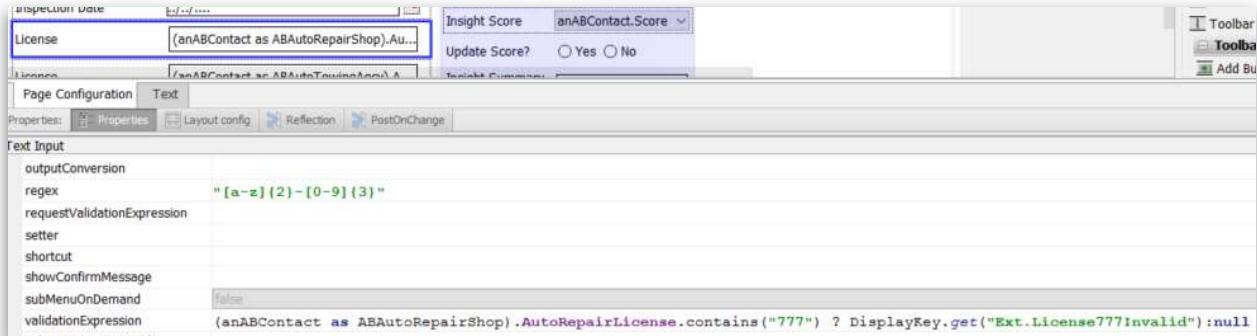
## 2. Configure the pattern

- Configure the regex property of the widget.



```
"[a-z]{2}-[0-9]{5}"
```

### 3. Configure the validationExpression property of the widget.



The screenshot shows the APEX page builder interface. A Text Input field is selected, and its properties are displayed in the 'Properties' tab. The 'validationExpression' property is set to the following JavaScript code:

```
(anABContact as ABAutoRepairShop).AutoRepairLicense.contains("777") ? DisplayKey.get("Ext.License777Invalid") :null
```

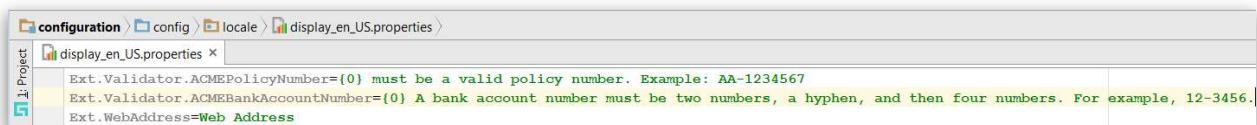
## 6.6 Lab Solution: Field-level validation in the Data Model



### Solution

Exact details on how to complete the lab.

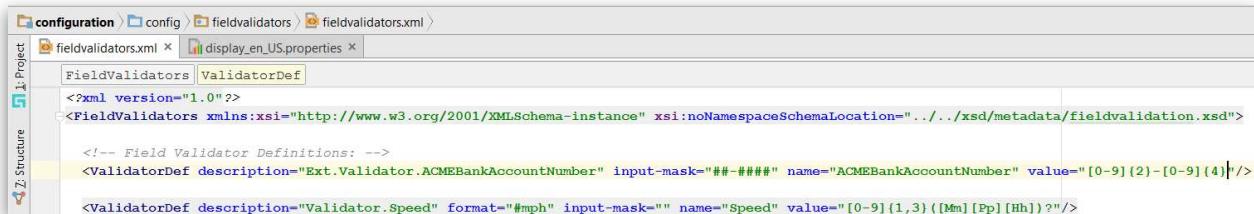
#### 1. Create the display key



The screenshot shows the APEX configuration interface. A new display key named 'display\_en\_US.properties' is being created under the 'configuration/config/locale' path. The content of the file is as follows:

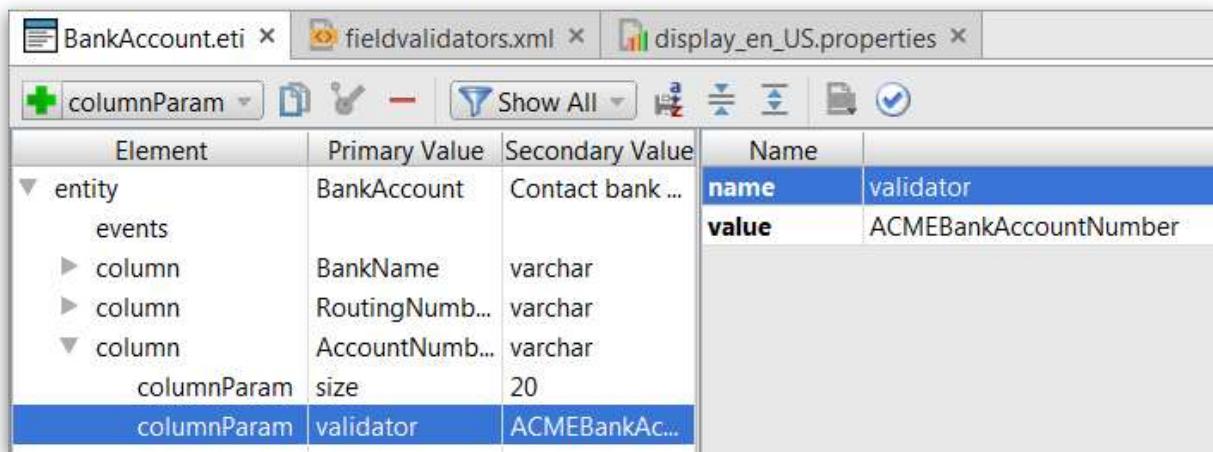
```
Ext.Validator.ACMEPolicyNumber={0} must be a valid policy number. Example: AA-1234567  
Ext.Validator.ACMEBankAccountNumber={0} A bank account number must be two numbers, a hyphen, and then four numbers. For example, 12-3456.  
Ext.WebAddress=Web Address
```

## 2. Create a field validator



```
<?xml version="1.0"?>
<FieldValidators xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../xsd/metadata/fieldvalidation.xsd">
    <!-- Field Validator Definitions: -->
    <ValidatorDef description="Ext.Validator.ACMEBankAccountNumber" input-mask="#-####" name="ACMEBankAccountNumber" value="[0-9]{2}-[0-9]{4}"/>
    <ValidatorDef description="Validator.Speed" format="#mph" input-mask="" name="Speed" value="[0-9]{1,3} ([Mm] ([Pp] [Hh])) ?"/>
```

## 3. Associate the field validator with the AccountNumber field of the BankAccount entity



Element	Primary Value	Secondary Value	Name	
entity	BankAccount	Contact bank ...	<b>name</b>	validator
events			<b>value</b>	ACMEBankAccountNumber
▶ column	BankName	varchar		
▶ column	RoutingNumb...	varchar		
▼ column	AccountNumb...	varchar		
columnParam	size	20		
columnParam	validator	ACMEBankAc...		

## 6.7 Lab Solution: Validation rules



### Solution

Exact details on how to complete the lab.

## 1. Create a new Rule in ABContactValidationRules.



```
ABContactValidationRules.grs x
└─ ABContact Validation Rules
    ├─ Default Validation Rules
    └─ Married Tax Filing Status

USES:
uses gw.api.locale.DisplayKey

CONDITION (aBContact : entity.ABContact):
return aBContact typeis ABPerson
and
(aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_JOINT
 or aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_SEPARATE
)
and
aBContact.MaritalStatus != MaritalStatus.TC_MARRIED

ACTION (aBContact : entity.ABContact, actions : gw.rules.Action):
    aBContact.rejectField("TaxFilingStatus",
        ValidationLevel.TC_LOADSAVE, DisplayKey.get("Ext.Validation.MarriedTaxFilingStatus"),
        null, null)
END
```

- a) You also have the option to copy and paste it from below. However, it is recommended that you type the code based on the screenshot. **Important:** please make sure that you copy and paste the sections individually without the parts highlighted in a darker gray.

```
USES:
uses gw.api.locale.DisplayKey

CONDITION (aBContact : entity.ABContact):
return aBContact typeis ABPerson
and
(aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_JOINT
 or aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_SEPARATE
)
and
aBContact.MaritalStatus != MaritalStatus.TC_MARRIED

ACTION (aBContact : entity.ABContact, actions : gw.rules.Action):
    aBContact.rejectField("TaxFilingStatus",
        ValidationLevel.TC_LOADSAVE,
        DisplayKey.get("Ext.Validation.MarriedTaxFilingStatus"),
        null, null)
END
```

## **2. Add the display key**

- a) Open the display.properties file
- b) Add a new entry as listed below:

```
Ext.Validation.MarriedTaxFilingStatus= Married filing jointly/Married filing separately is only  
allowed when contact is married.
```