FAQ - Business Rules

Business Rules have several quirky behaviors that will be addressed in a future release. Please read and understand these points before writing your first business rule.

Why do I need a local variable for certain cases ?

There are rules scenarios where you can't use a Live Forms control directly in a rule expression and you are forced to define a local variable, perform the computation using the local variable and assign the value to the control at the conclusion of the computation.

For instance, assume a form that calculates a total from a list of subtotals. The total is a control T and subtotals are represented as a repeat of controls S. In other words, T holds the total of adding all values in S. You could write the rules as:

```
T.value = 0;
for (var i = 0; i < S.value.length; i++) {
    T.value = T.value + S[i].value;
}
```

The code above makes logical sense but it just won't work. In a nutshell this particularity of the Live Forms implementation can be explained this way:

*The right hand side of an assignment expression resolves to **the value** of a Live Forms control passed to a rule when it is invoked. The left hand side uses a reference to the control'.*

Lets work with a concrete example to understand the concept. Assume that there are two subtotals in the form, 12 and 14 and the total is correctly set to 26. Now, you add 50 as a third subtotal and the rule fires as a consequence. The values of the controls involved in the rule are passed as parameters to the rule: the subtotals 12,14 and 50 and the current total (value of T), 26. The core of the matter is in the expression:

```
T.value = T.value + S[i].value;
```

The expression will be evaluated three times (since there are three subtotals) but for every iteration, the *T.value* operand on the right hand side will always evaluate to 26 which is not the desired behavior and will lead to an incorrect result. Again, this happens because the right hand side of that expression is resolved based on **the values** initially passed to the rule as parameters.

Live Forms explicitly imposes this restriction for the sake of efficiency and there would be a significant impact in performance if the restriction was not in place. Lets rewrite the rule using a local variable:

```
var tot = 0;
for (var i = 0; i < S.value.length; i++) {
    tot = tot + S[i].value;
}
T.value = tot;
```

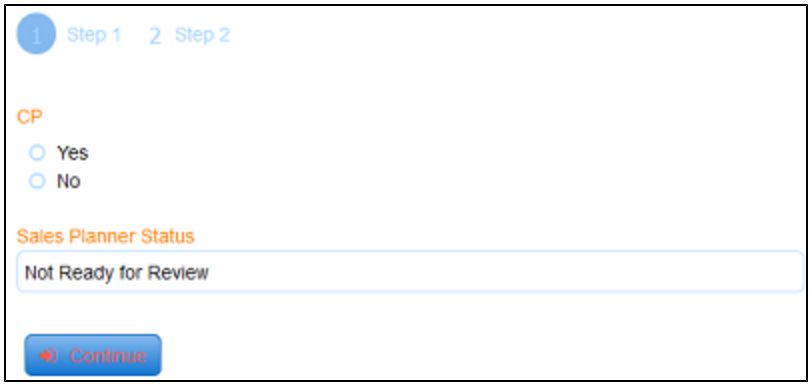
This rule will produce the expected result.

> Variables in rules must be unique. Having duplicate variables in the same rule will cause the rule to fail but you may not get an error message.

Why do my rules show unexpected results in a flow using Linked activities?

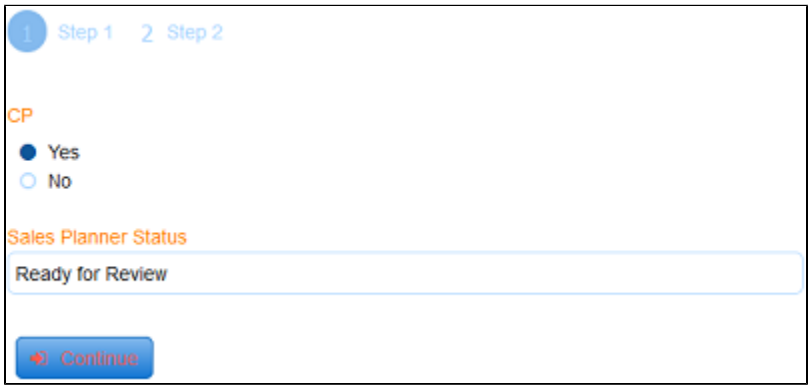
This question is best answered with an example. Let's say you have a two step flow that contains the following rules; When step 1 of the flow is execute, the form loads and the Sales Planner Status control displays "Not Ready for Review".

```
Sales Planner Status:
if (form.load) {
   SalesPlannerStatus.value = "Not Ready for Review";
}
```

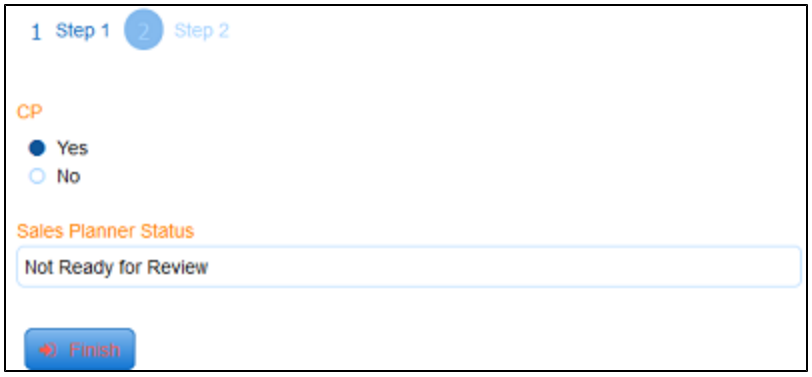



```
CP Status:
if (CP.value == "Yes") {
  SalesPlannerStatus.value = "Ready for Review";
} else if (CP.value == "No") {
  SalesPlannerStatus.value = "Not Ready for Review";
}
```

If the user clicks Yes in the CP field, the Sales Planner Status control displays "Ready for Review"

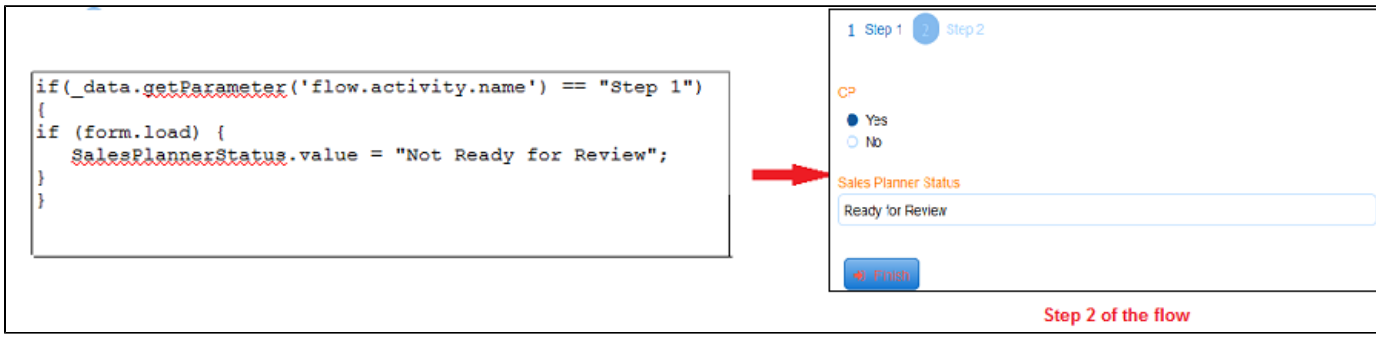


When you click Continue to advance to the next step of the flow, the value of the Sales Planner Status is set to "Not Ready for Review" even though value of CP is still Yes.



This is as designed. The logic purposely ignores setting value properties that happen in a flow during form state initialization. This is what happens in the second step of this flow using linked activities. Note that changes to other attributes other than value are not ignored during initialization. The designer will see a message in the log and debug console output that says the set of the property value is ignored.

Adding if (_data.getParameter('flow.activity.name') == "Step 1") to the Sales Planner Status rule, gives the expected results:

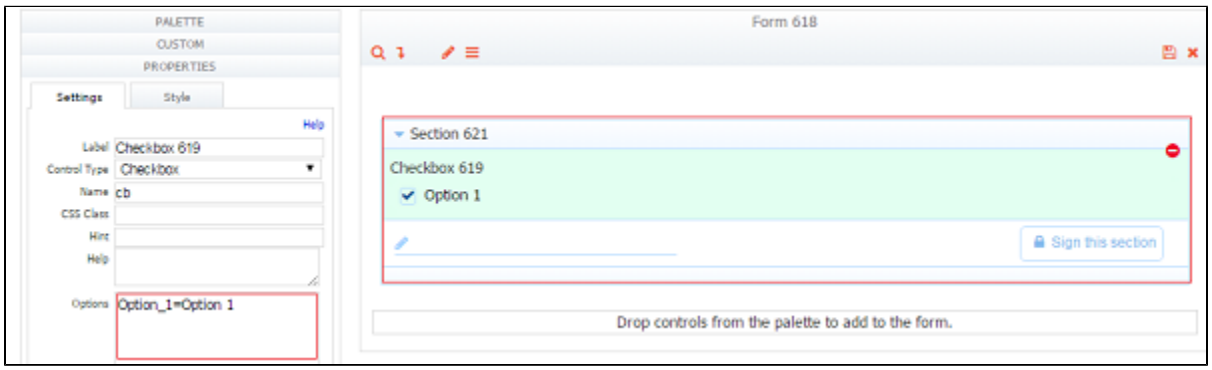


What causes an "Invalid signature detected. Data may have been tampered with" error?

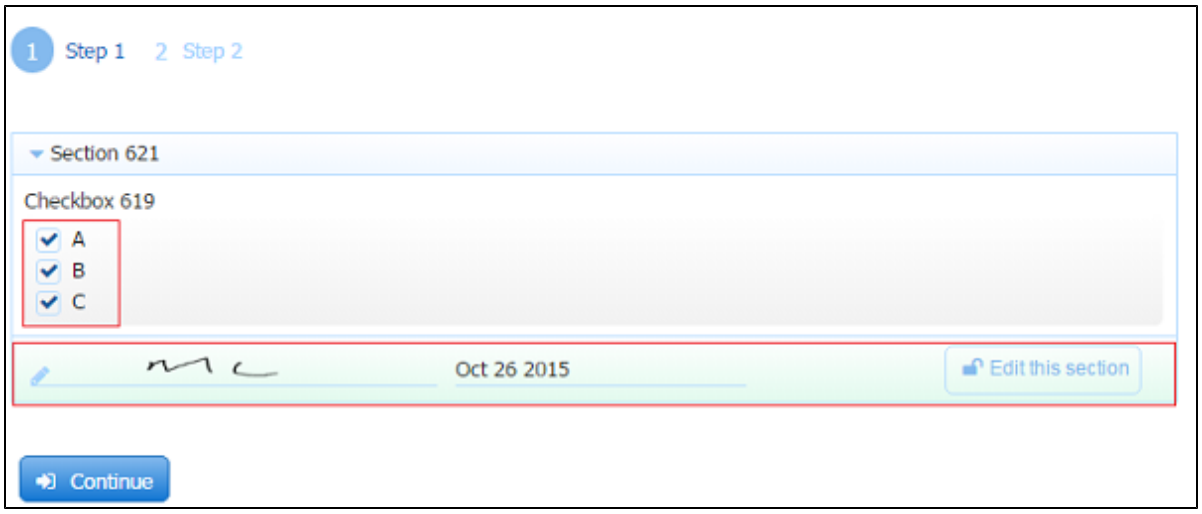
This error message usually displays in multi-step workflows when there is a rule on a subsequent step that executes at form load and updates values inside a signed section in a previous step. Here are some examples and solutions:

Checkbox with Custom Options in a locked Signed Section

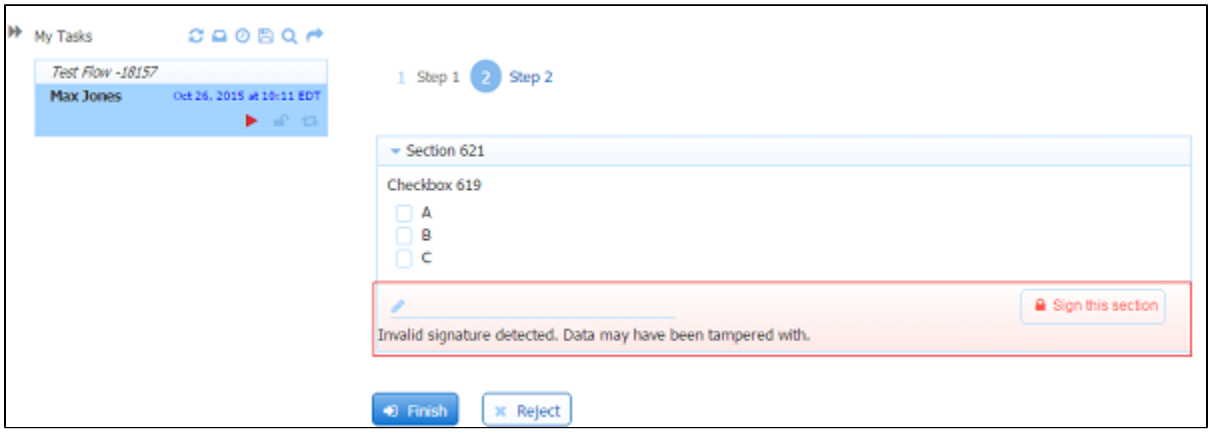
Imagine a two step workflow designed with Linked Steps where step 1 contains a Checkbox control inside a section that the user is required to sign, The checkbox has only one option defined in the flow designer. A Business rule is executed when step 1 loads to dynamically populate the checkbox with additional options.



The image shows this scenario in use mode after the rule has executed. Notice there are three options for the checkbox now. The user selects all three options, signs then clicks continue.

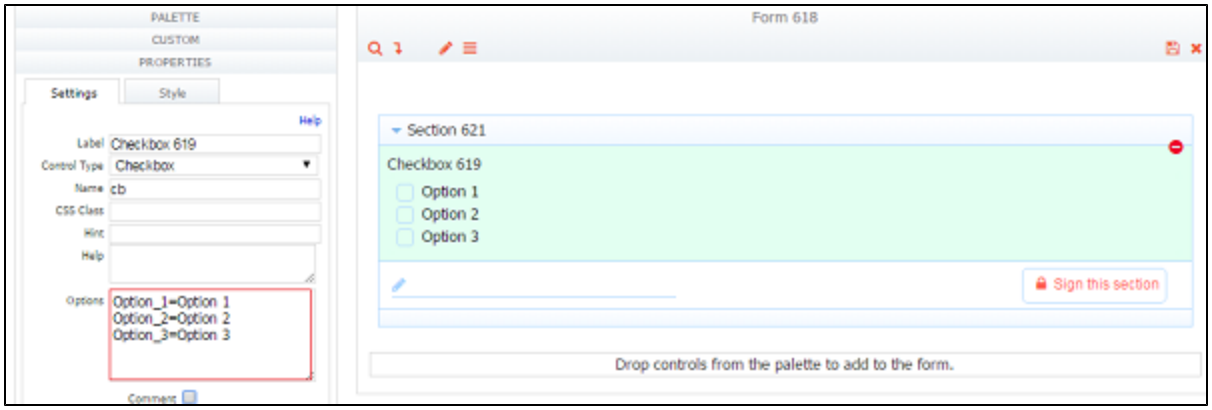


The flow is routed to Tasks Lists of users who fulfill the requirements for the next step. When Step 2 loads, the rule is executed again. This combination results in data tampered error.



Solution:

This situation can be avoided by always setting more than one option in the flow designer for any checkbox that will be dynamically populated with a rule.



Changing the Value of Controls in a Signed Section

Business rules that change the values of controls inside a signed section may result in the "Invalid signature detected. Data may have been tampered with." error. Rules that "tamper with the data" and invalidate a signature once a section has been signed are no longer allowed. Refer to this topic for the details.

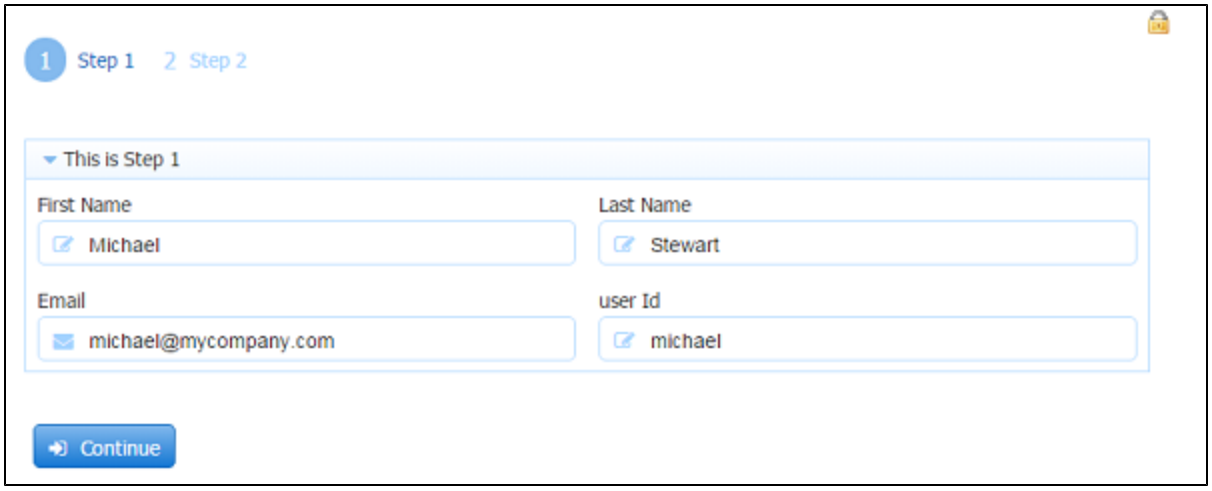
Why does the information for the logged in user change when I navigate forward/backwards in my workflow?

Designers can use a Business Rule to initialize fields in the first step of a flow with information about the logged in user. If your flow navigates to a different user/role for subsequent steps, and the user navigates back to step 1, the rule executes again when Step 1 loads, overwriting the information in step 1 with the information of the user performing the subsequent step. This is an important consideration whether your workflow is designed using individual or Linked Steps.

You can improve your rule to account for this. For example, let's say you have a two step flow. Step 1 executes this rule to fill in the First Name, Last Name, Email and the Live Forms user id into fields named First Name, Last Name, Email and UserID respectively. Note this rule executes when Step 1 loads.

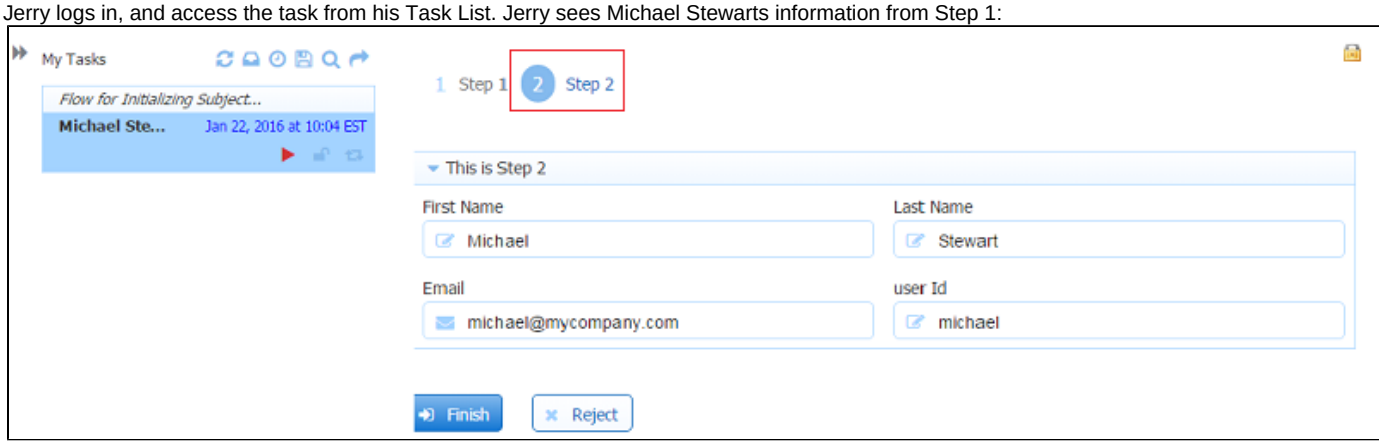
```
if (form.load) {
    FirstName.value = _data.getParameter('subject.first.name');
    LastName.value = _data.getParameter('subject.last.name');
    Email.value = _data.getParameter('subject.email');
    UserId.value = _data.getParameter('subject.id');
  }
```

When user Michael Stewart executes the first step of the flow, here is what you see:



Michael clicks continue and the flow progresses to Step 2 which is designed to be performed by a user with the Manager role, Jerry.

Jerry logs in, and access the task from his Task List. Jerry sees Michael Stewarts information from Step 1:

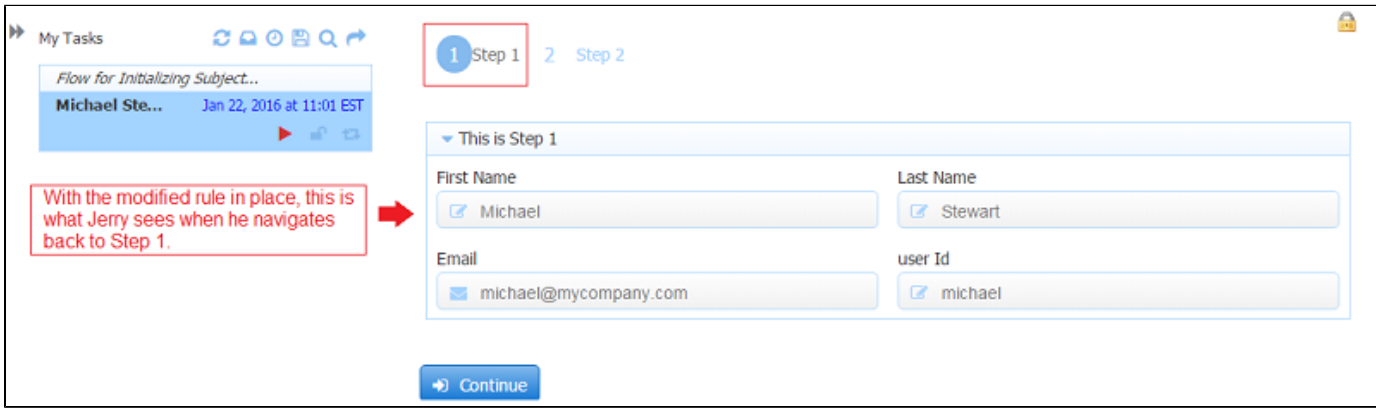


However, if Jerry navigates back to Step 1, the rule executes again when Step 1 loads and overwrites Michael's information with Jerry's:

To prevent initialized fields from being overwritten if users navigate to a previous step in a flow, you must add a condition to your rule to only initialize the fields if they have not already been initialized when the step loads. Adding the second line to the rule used in this example ensures that initialized fields will not be overwritten.

```
if (form.load) {
    if (FirstName.value.length === 0) {
    FirstName.value = _data.getParameter('subject.first.name');
    LastName.value = _data.getParameter('subject.last.name');
    Email.value = _data.getParameter('subject.email');
    UserId.value = _data.getParameter('subject.id');
  }
}
```

The image shows what Jerry sees when he navigates back to Step 1 with the modified rule in place:



You can adapt the conditional statement in this example to fit your rule.