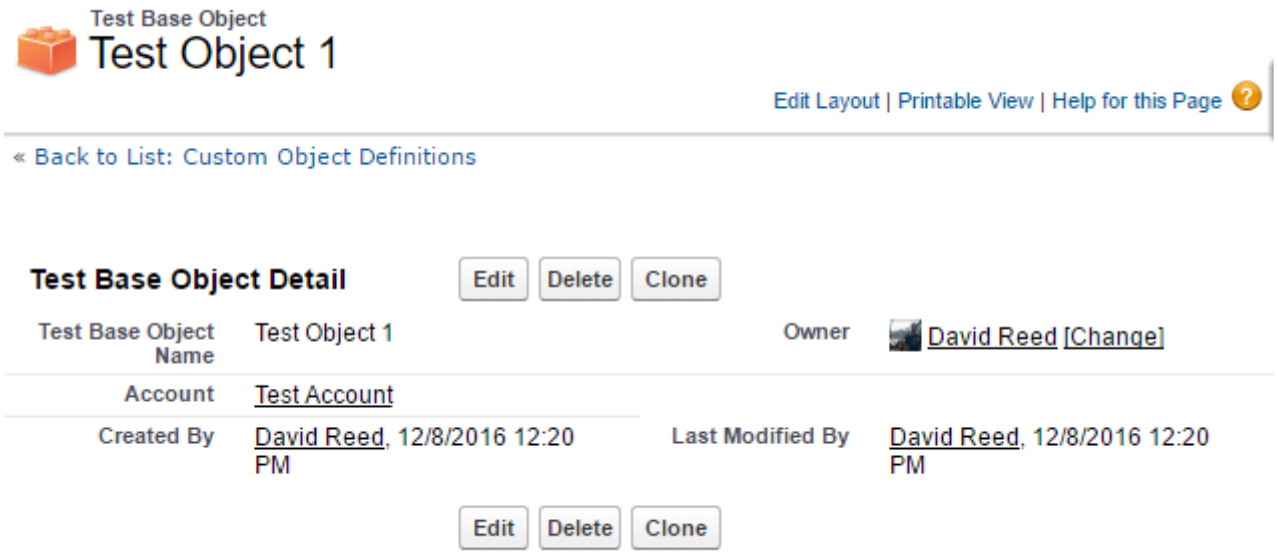
Null Relationships and Short-Circuiting Behavior in Salesforce Formulas, Process Builder, Flow, and Apex

08 Dec 2016

What happens when you refer to a field across a lookup relationship, and the lookup relationship is `null`? The answer turns out to vary across contexts in non-obvious ways. In the course of debugging some Process Builder logic, I came up with a summary. In all of the examples below, I'm using a custom object called `Test Base Object` with a nullable lookup relationship `Account__c`.



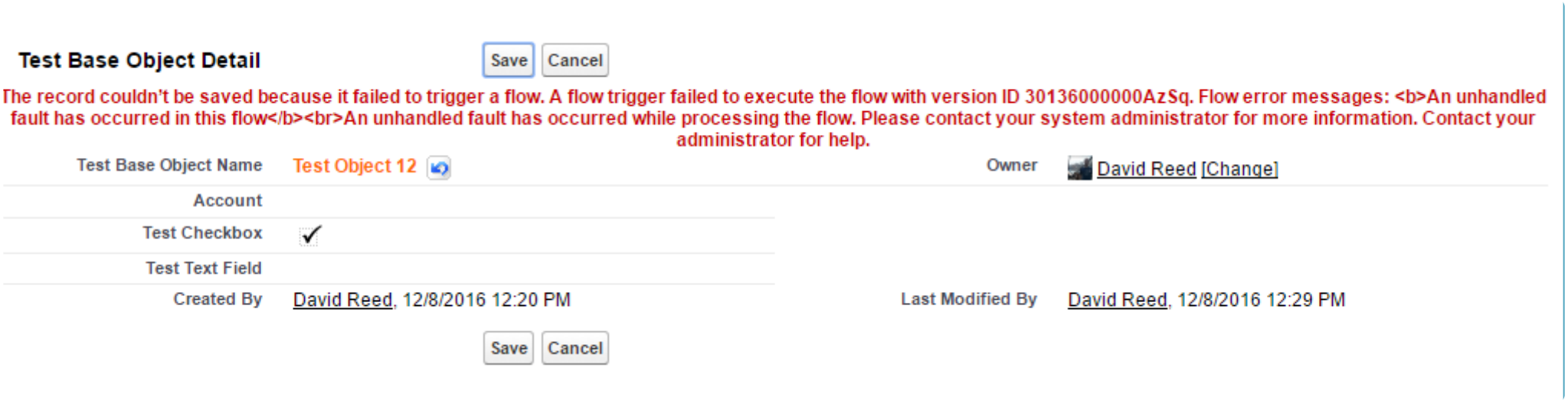
Formula Fields

When a cross-object reference is used in a formula field and the lookup is `null`, the value of the cross-object reference is also `null`. No exception is thrown.

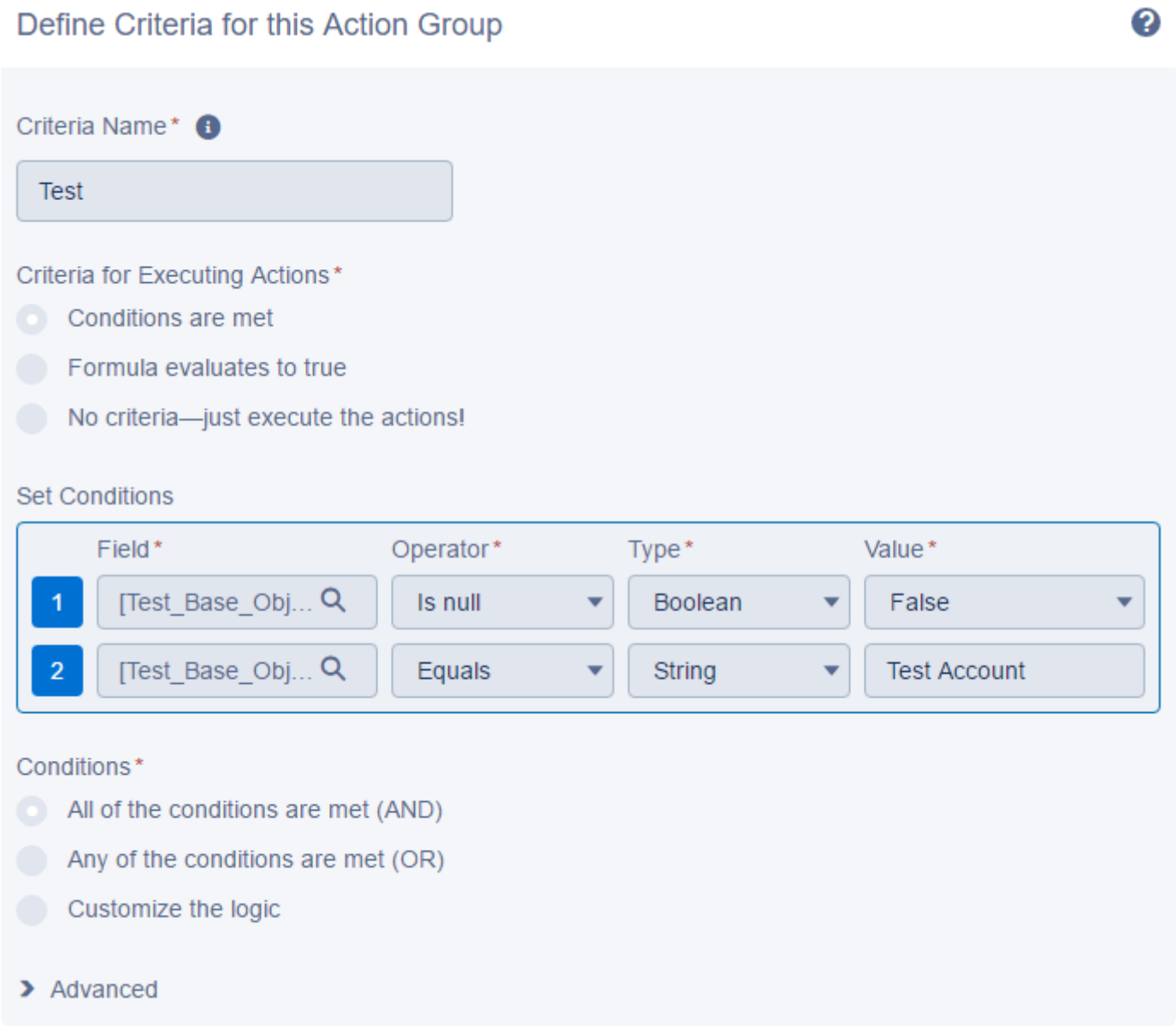
Boolean logic treats the `null` value as `false`. Since no exception occurs, the ordering of Boolean clauses is irrelevant and no short-circuit evaluation is needed to obtain correct results.

Process Builder

Process Builder's condition and formula-based triggers *appear* to operate like formulas, but actually handle `null` relationships very differently. *Dereferencing a* `null` *field in a Process Builder condition or formula always results in an exception*. With complex logic in conditions for running actions, this can be tricky to debug. The errors it produces for users are opaque and frustrating, often preventing any mutation of the involved object.



Fortunately, Boolean operators and functions (`AND` and `OR`, including the implicit logical operations used in condition-based triggers) in the Process Builder context perform *short-circuit evaluation*. In other words, references across the lookup relationship can be guarded by checks against `null` lookups earlier in the evaluation order such that evaluation will stop *before*reaching the cross-object relationship, avoiding an exception. The evaluation order is left-to-right for the `AND()` and `OR()` functions and the `&&` and `||` operators, and top-to-bottom for condition lists.



Using conditions in Process Builder, always precede a cross-object field reference (assuming a nullable lookup relationship) with a null check. As in this example, protect the `[Test_Base_Object__c].Account__c.Name` reference with a preceding "Is null" condition on `[Test_Base_Object__c].Account__c`. Because the criteria are set to require "All of the conditions

are met (AND)", if Condition 1 evaluates to `false` (indicating a `null` value in the lookup field), evaluation of conditions will immediately stop, and no exception will be thrown.

Note that this won't work the same way using an OR condition. OR short-circuits on `true` values, and short-circuiting because of a `null` lookup is often not the desired behavior. In many cases, it's easier to handle possible `null` relationships by using customized logic and nesting an AND with the above null-check within the OR. Constructing a formula may be more straightforward.

Formulas in Process Builder short-circuit in the same way, whether using the `AND()` and `OR()` functions or the `&&` and `||` operators. The following pattern is safe.

Define Criteria for this Action Group

Criteria Name *

Test

Criteria for Executing Actions *

- Conditions are met
- Formula evaluates to true
- No criteria—just execute the actions!

Build Formula

```
AND(NOT(ISBLANK([Test_Base_Object__c].Account__c)),
[Test_Base_Object__c].Account__c.Name = 'Test Account')
```

Advanced

Flow

sObject variables in Flow present a challenge. While sObject variables can be `null`, and cross-object field references that traverse a `null` variable *will* result in an exception being thrown, one cannot directly test an sObject variable's nullity within a Flow formula. `ISBLANK(sObjectVariable)` and `ISNULL(sObjectVariable)` aren't legal and will prevent your Flow from being activated.

There are a couple of ways to work around this limitation.

One is to check the sObject variable's nullity using a Decision element before using any formulas that references its fields. (See the release notes on cross-object references in Flow). Unfortunately, this may not be practicable in a flow where formulas make complex decisions or calculate across a number of different objects.

Another option, if the variable is populated using a lookup element from a given Id value, is to check the nullity of the Id value field in the formula that performs the cross-object reference. Like in Process Builder, logical functions in Flow formulas use short-circuit evaluation. This allows you to effectively guard cross-object references against nulls in the circumstance that the potentially-null sObject variable is looked up from an Id field, rather than other criteria.

Finally, as discussed in the Summer '14 release notes, you can provide a fault path. While this offers less of an opportunity to handle decision-making or conditional data within a single formula, it permits clearly expressing error- and null-handling within the logic of the flow itself.

Apex

In most cases, Apex handles `null` relationships in the same way Process Builder formulas do. However, there's one variant case: the code below does not crash.

Enter Apex Code

```
Test_Base_Object__c t = [SELECT Account__r.Type
                         FROM Test_Base_Object__c
                         WHERE Name = 'Test Object 1'];

System.debug(t);
System.debug(t.Account__c);
System.debug(t.Account__r.Type);
```

Open Log | Execute | Execute Highlighted

Accessing the relationship path directly from the queried object simply results in a `null`; no exception is thrown.

However, this only works when you traverse the relationship via the queried sObject. If the intermediate object value (which is `null`) is assigned to another variable before dereferencing its field, you get a `NullPointerException` .

Like Process Builder formulas, Apex supports short-circuit evaluation. The code below outputs `null` , `null` , and `false` before finally throwing an exception at line 10.

Enter Apex Code

```
1  Test_Base_Object__c t = [SELECT Account__r.Type
2                           FROM Test_Base_Object__c
3                           LIMIT 1];
4
5  Account a = t.Account__r;
6
7  System.debug(a);
8  System.debug(t.Account__r.Type);
9  System.debug(t.Account__c != null && a.Type == 'Other');
10 System.debug(a.Type);
```

Open Log | Execute | Execute Highlighted