Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Advanced
Works with: Lotus Workflow 3.0
Updated: 01-May-2002

by Hans Boone
and Cees van der Woude

Lotus Workflow has brought new power and flexibility to process improvement and process management. For example, Lotus Workflow can automate your corporate approval processes, quickly and efficiently routing requests as they progress from one person in the approval chain to the next. Process designers can use the intuitive interface in the Lotus Workflow Architect to construct approval and other business processes, reusing modular business objects and assigning essential business activities to individuals and groups through a company's personnel directory. Application development is straightforward for any developer who has used Domino Designer, and with the release of Lotus Workflow 3.0, application design has become easier than ever.

Although Lotus Workflow 3.0 is superbly flexible "out of the box," it is even more flexible with the use of LotusScript, JavaScript, and formulas. This article shows how to use these programmatic capabilities to:

- Perform field-level validation using the Lotus Workflow Architect
- Create a decision activity that loops back on itself

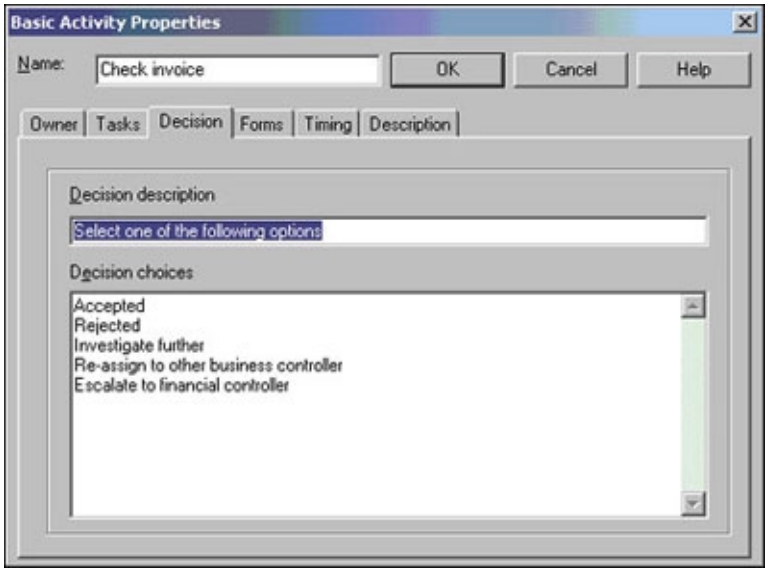
To demonstrate these features, we'll show how they can enhance an automated approval process. In our first example, we'll use field-level validation to require users to enter an explanation when they reject a request. And in the second, we'll create a decision activity loop that completes the activity as soon as a certain number of activity owners give their approval.

This article assumes you are an experienced LotusScript and JavaScript programmer and are familiar with Lotus Workflow and Domino Designer. It is based on several developer tips published on the Lotus Workflow page. For the latest information on Lotus Workflow (including other tips and resources), see the **Lotus Workflow Home page**.

Field validation using the Lotus Workflow Architect

Imagine your organization needs to set up an invoice approval process where, if an activity owner rejects an invoice, you want to require them to enter an explanation. (In other words, you want to validate field values based on a specific decision.) Further, your solution must work equally well for Web browsers and the Notes client. And if you can, you want to code this business rule in the process definition using the Lotus Workflow Architect.

To begin, let's create an activity called Check invoice, where several decisions are configured, for example, Accepted, Rejected, Investigate further, Reassign to other business controller, and Escalate to financial controller.



If the activity owner selects Rejected and doesn't enter an explanation in the CommentLine field, we want to generate an error. Because we want this to work for both a Notes client and a Web browser, we'll create a LotusScript solution for Notes and a JavaScript solution for the Web.

Setting up field validation for the Notes client
First, we'll create LotusScript code for validating field values (business rules) in the process definition using the Lotus Workflow Architect. This consists of two basic steps:

1. Create a custom attribute to define a static validation condition.
2. Change the QueryActivityCompleted_ event to evaluate this condition at runtime.

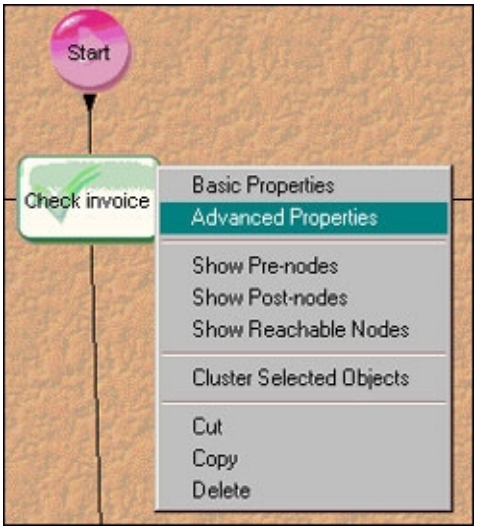
Creating a custom attribute
The Check invoice activity needs a custom attribute, RequiredFieldNotes, with the following formula:

"ApprovalChoiceOS='Rejected' & CommentLine="%You have to supply a reason for rejection"

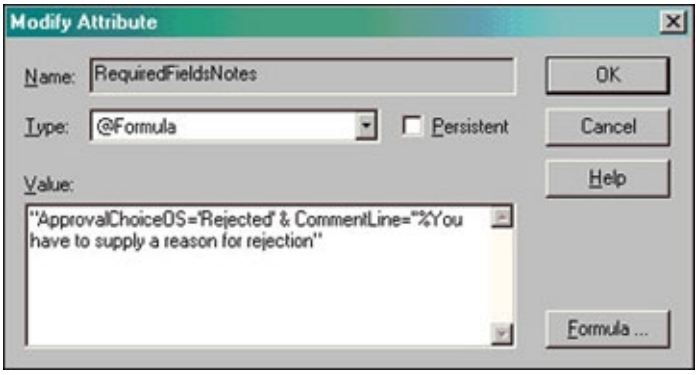
In the formula, ApprovalChoiceOS is the field used to store a decision of a certain activity. This field is created by the Lotus Workflow system. CommentLine is a custom field on the main Lotus Workflow form used to collect comments for each activity. (You can create the main Lotus Workflow form by creating and changing a copy of the Sample Form in the Lotus Workflow application template. You can then add the CommentLine field to this form, using Domino Designer.)

To create the custom attribute for the Check invoice activity:

1. Right click the Check invoice activity and select Advanced Properties.



2. Click on the Custom tab of the properties box and select New.
3. Type the name of the new custom attribute, RequiredFieldsNotes, in the Modify Attribute dialog box.
4. Select @Formula for Type and enter the formula listed above in the Value text box.



The syntax for this custom attribute is:

"<Condition for failing validation>%<Validation failed message>"

Note that if you wanted to check for multiple conditions, you would use the standard list separator (" **:**") between conditions:

"<Condition 1 for failing validation>%<Validation failed message 1>":"<Condition 2 for failing validation>%<Validation failed message 2>"

Changing the QueryActivityCompleted_ event
Next, we need to add code to the QueryActivityCompleted_ event in the OS Application Events script library. This code will check whether the condition <Condition for failing validation> is true and if so, generate an error and display the <Validation failed message>.

To do this, enter the following LotusScript code into the QueryActivityCompleted_ event (you can copy and paste this sample if you like). The code will break up the RequiredFieldNotes custom attribute into the validation condition and the error message. The validation condition is evaluated against the Main Lotus Workflow form. If validation fails, the error message is displayed using a message box.

```
Private Function QueryActivityCompleted_(Continue As Integer, CoverDocument As NotesDocument,
BinderDocument As NotesDocument, Uiws As Variant, ErrorCode As Integer, FailureList As Variant, sUserName
As String)
Dim db As notesdatabase
```



```
Dim MainDocument As notesdocument
Dim RequiredFields As Variant
Dim ValidationFailed As Variant
Dim ValidationCondition As Variant
Dim ErrorMessage As Variant
Dim i As Integer
Dim Answer As Integer
Dim Debugging As Integer

On Error Goto errortrap
Debugging = False

Set db=CoverDocument.Parentdatabase
Set MainDocument = db.GetDocumentByUNID(CoverDocument.MainDocIDOS(0))

If Not MainDocument Is Nothing Then
    RequiredFields = MainDocument.GetItemValue("RequiredFieldsNotes")

    ' Use created custom attribute RequiredFieldsNotes, this attribute is available on the Main document as
    ' an normal Notes field.
    If RequiredFields(0) = "" Then
        If debugging Then Messagebox "no required fields - empty set"
        Continue = True
    Else
        ' Loop through each of the possible multiple conditions
        For i = 0 To Ubound (RequiredFields)

            ' Break up the RequiredFieldNotes custom attribute in the validation condition and the error message
            ValidationCondition = Evaluate(|@Word("|+RequiredFields(i)+|";"%";1)|)
            ErrorMessage = Evaluate(|@Word("|+RequiredFields(i)+|";"%";2)|)

            ' Evaluate the validation condition
            ValidationFailed = Evaluate( |@if(| + ValidationCondition(0) + |;"true";"false")|, MainDocument)

            ' Display debugging messages in debugging variable is set to true
            If debugging Then Messagebox ValidationCondition(0)
            If debugging Then Messagebox ValidationFailed(0)
            If debugging Then Messagebox ErrorMessage(0)

            ' Check if validation failed and if so display the error message
            If ValidationFailed(0) = "true" Then 'if true, validation failed and we've to display a message
                Answer = Msgbox (ErrorMessage(0), 16, "Validation error") 'display the <Validation failed message>
                Continue = False
                Goto ExitFunction
            End If
        Next
    End If
Else
    Messagebox("Main document not found")
End If
Goto exitfunction

ErrorTrap:
If debugging Then Messagebox "Error: " + Cstr(Err)+ " " + Error(Err)
Continue=False
Resume ExitFunction

ExitFunction:
If Continue = False Then
    ' Validation failed, open the main document again
    Call uiws.EditDocument(True, MainDocument)
End If
End Function
```


For debugging purposes, you can set the variable debugging to true. Then you'll see message boxes for:

- Validation condition
- Evaluated validation condition (True or False)
- Error message

Validating field values using a browser
As you can see, validating field values for the Notes client is fairly straightforward. Validating field values using a Web browser is a little more complicated in that it requires JavaScript code. The underlying logic, however, is the same. To validate field values using a browser we need to code the following steps:

1. Create a custom attribute to define a static validation condition.
2. Add the custom JavaScript function CustomActivityValidation to the JS Header section of the OS Domino Workflow Web subform. This function is called when a user completes the Check invoice activity (submits the Lotus Workflow form).
3. Add the custom JavaScript function Blank to the JS Header section of the OS Domino Workflow Web subform. This JavaScript function is used to perform the actual field validation.
4. Add the GetErrorMessage JavaScript function to the OS Domino Workflow Web subform to evaluate the static validation condition (as defined in step 1) at runtime. If the validation failed, this function will return the error message defined in step 1.

Creating the custom attribute
For the Check invoice activity, create another custom attribute, RequiredFieldWeb, with this formula:

```
"LWFUtilGetFormFieldValue(WFDoc.Form.ApprovalChoiceOS) == 'Rejected' &
Blank(WFDoc.Form.CommentLine)%You have to supply a reason for rejection"
```

Once again the syntax for the custom attribute is:

```
"<Condition 1 for failing validation>%<Validation failed message 1>":"<Condition 2 for failing
validation>%<Validation failed message 2>"
```

The conditions have to be valid JavaScript statements, evaluating to true or false.

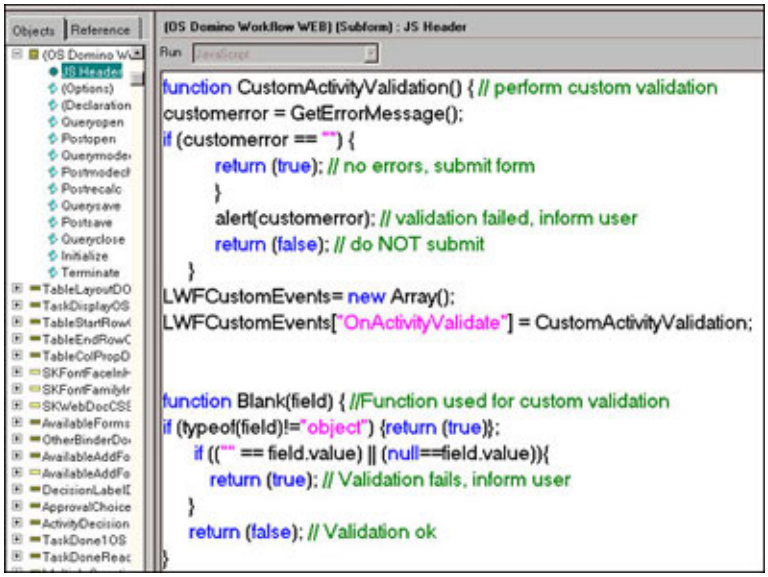
Adding the custom JavaScript functions to the JS Header section
Next, add the custom JavaScript function, CustomActivityValidation, to the JS Header section of the OS Domino Workflow Web subform. To do this, you add the following script at the end of the JS Header section of the OS Domino Workflow Web subform:

```
function CustomActivityValidation() { // perform custom validation
customerror = GetErrorMessage();
if (customerror == "") {
return (true); // no errors, submit form
}
alert(customerror); // validation failed, inform user
return (false); // do NOT submit
}
LWFCustomEvents= new Array();
LWFCustomEvents["OnActivityValidate"] = CustomActivityValidation;
```

Add another custom JavaScript function, Blank, to the JS Header section of the OS Domino Workflow Web subform. To do this, you add the following script at the end of the JS Header section of the OS Domino Workflow Web subform:

```
function Blank(field) { //Function used for custom validation
if (typeof(field)!="object") {return (true)};
if (("" == field.value) || (null==field.value)){
return (true); // Validation fails, inform user
}
return (false); // Validation ok
}
```


The end of the JS Header section should look like this:



Your validation code will now automatically be encapsulated in WFDoc.submit. This offers the advantage of not needing to modify the Workflow JavaScriptAPI page. Adding your code to the OS Domino Workflow Web subform is better than modifying the Workflow JavaScriptAPI page, because the OS Domino Workflow Web subform is part of the Skin. The Skin can be modified by designers to customize the Lotus Workflow Web look-and-feel. The function CustomActivityValidation will be executed by the Lotus Workflow ExecuteEvent function on the Workflow JavaScriptAPI page when a user completes the activity (submits the Lotus Workflow form).

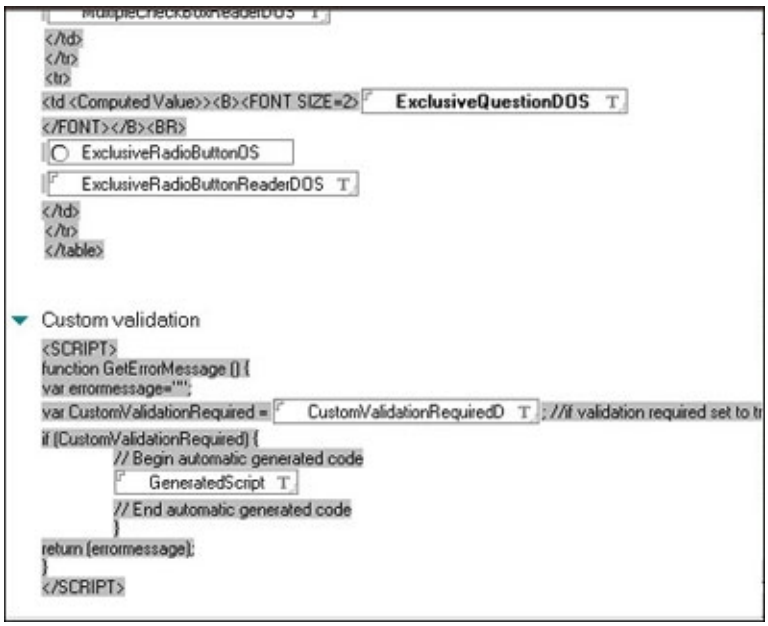
The function Blank is used to determine whether a field on the form contains a value. If required, add other custom functions; for example, you might add validation for date format checking, greater than, and so on.

Adding the GetErrorMessage JavaScript function

Next, you need to add the GetErrorMessage JavaScript function to the body of the OS Domino Workflow Web subform. If a user tries to complete the Check Invoice activity, the CustomActivityValidation JavaScript function is called. This function, in turn, will call the GetErrorMessage JavaScript function; and this function will return an error message if the validation failed.

Add the following script to the body of the OS Domino Workflow Web subform, formatting it for pass-thru HTML:

```
<SCRIPT>
function GetErrorMessage () {
var errormessage="";
var CustomValidationRequired = [CustomValidationRequiredD]; //if validation required set to true
if (CustomValidationRequired) {
// Begin automatic generated code
[GeneratedScript]
// End automatic generated code
}
return (errormessage);
}
</SCRIPT>
```

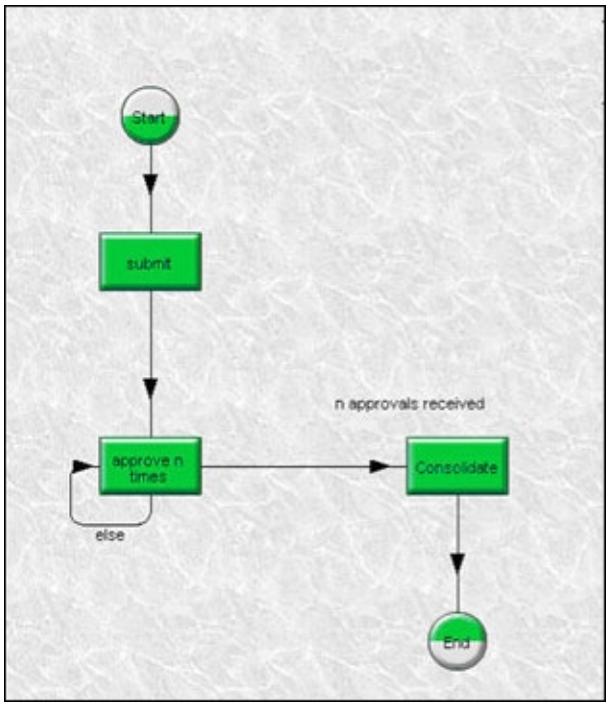
The two fields in this script should be created in the body of the subform as follows:

Field name	Field properties	Field formula
[GeneratedScript]	text, multi-value, display / separate values with New Line, computed for display	Temp := RequiredFieldsWeb; Condition := @Word(Temp;"%";1); Message := "\""+@Word(Temp;"%";2) + "\""; Script := "if ("+Condition+") { var errormessage = " + Message +";};"; @If(Temp = "";"// No validation required";Script)
[CustomvalidationRequiredD]	text, computed for display	@If(RequiredFieldsWeb !="";"true";"false"); REM "RequiredFieldsWEB is a custom attribute set in the Architect";

Test your application in the Web browser and the Notes client. If you need to debug your Web application, consider using Netscape. If you type "javascript: <enter>," you can see generated JavaScript errors.

Creating a decision activity that loops

Let's say you wanted to set up an approval process where the process can only go forward to the next activity after (for example) three people in a particular group of five indicate approval. The logic flow might look something like this:



To do this, you can use the new Lotus Workflow 3.0 loop-back-onto-itself option in the Architect, and three @JobProperty functions.

Creating the @JobProperty functions
As you probably know, @JobProperty functions take runtime information—that is, information we do not know at design time—and evaluate it. The @JobProperty function you're likely to use most often is @JobProperty[Initiator], which can look up the name of the initiator once a job starts.

In our example, we'll create three @JobProperty functions called @JobProperty[Number of Decisions], @JobProperty[Previous Activity Owners of this Activity], and @JobProperty[No More Participants]. These functions are more complex than @JobProperty[Initiator] but basically look at information gathered in the ActivityLogOS multi-value field on the cover document.

Note: When creating the following formulas, make sure you don't put any line breaks in them.

Number of decisions
The @JobProperty[Number of Decisions] function determines the number of decisions that have been made during this job. Its syntax is:

```
t1:=(@Word(@GetDocField(FolderIDOS;"ActivityLogOS");"#";7))+(@Word(@GetDocField(FolderIDOS;"ActivityLogOS");"#";1));t2:=@Trim(@Replace(t1;@Input+ActivityIDOS;""));@Elements(t1)-@Elements(t2)
```

where t1 gets a list of all previous decisions and associated activityIDs and t2 extracts the relevant decisions and the activityID from the list t1.

After you create the @JobProperty[Number of Decisions] function, it will be listed in the Architect library. You can select this function by double-clicking it. The function will appear in the formula pane, where you can work with it like other @functions. Note that @JobProperty[Number of Decisions] requires the parameter @Input, which you specify in the Architect. Finally, it subtracts the number of elements in the second list from the original list.

Previous activity owners of this activity
The @JobProperty[Previous Activity Owners of this Activity] function creates a list of the activity owners who have participated in this activity up to this point. Its syntax is:

```
t1:=@Word(@GetDocField(FolderIDOS;"ActivityLogOS");"#";1);t2:=@Trim(@Replace(t1;ActivityIDOS;""));t3:=@Elements(t1)-@Elements(t2);@If(t3<>0;@Subset(@Word(@GetDocField(FolderIDOS;"ActivityLogOS");"#";2);-(t3));"
```



```
")
```

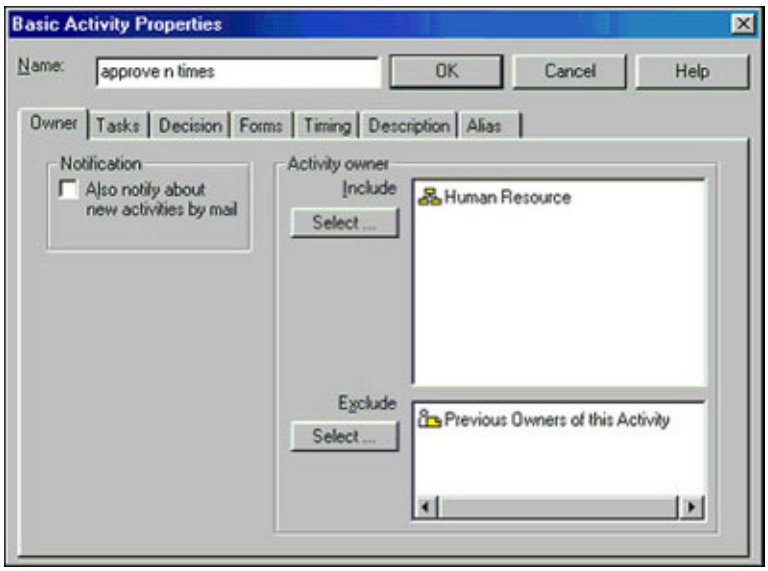
where t1 gets a list of all activity IDs (even those activities that preceded our looping activity), t2 removes all entries in this list that match the ID for our looping activity, and t3 calculates the difference in number of elements (n).

No more participants
The @JobProperty[No More Participants] function determines whether there are any more potential activity owners left for this activity. Its syntax is:

```
@Elements(ParticipantOS)=1
```

Creating the process
Create a simple process with a Submit, Approve n times, and Consolidate activity, as illustrated in the previous screen. The second activity in your process, Approve n times, gathers the decisions, so you must define a decision for this activity such as "Please decide"; Options: Approved, Rejected .

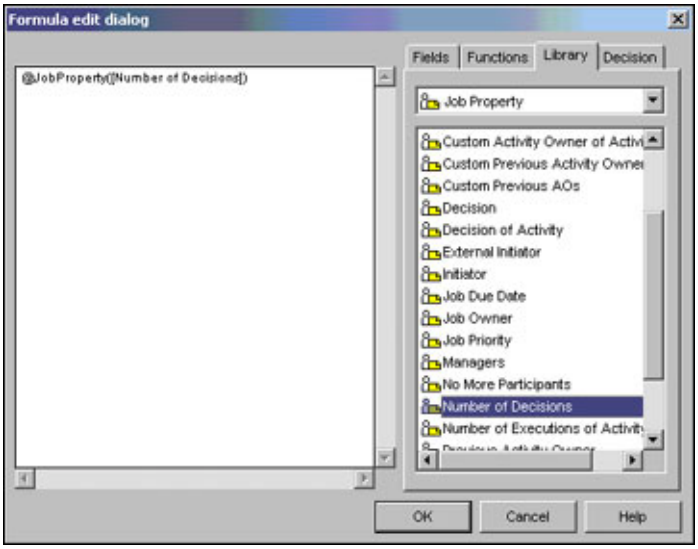
The owners of the Approve n times activity can be selected as usual: a group, role, department, or formula. In the Exclude list of the Owner tab, select the @JobProperty[Previous Owners of this Activity] function that you created.



This Exclude list selection ensures that anyone on the Include list (in this case, anyone in Human Resources) can make a decision only once.

The arrow that feeds into the Consolidate activity should be type Condition. To specify the formula for the condition:

1. Click Formula to see the Formula edit dialog box.
2. On the Library tab, select Job Properties and then locate Number of Decisions in the list and double-click it. The selected function will appear on the left-hand side of the Formula edit dialog box.



3. To complete the function, position the cursor before the closing parenthesis, type a semicolon (;) and then enter the decision option you are checking for. In this example, we offer the decision options Approved and Rejected and check for n approvals. The completed function should read:

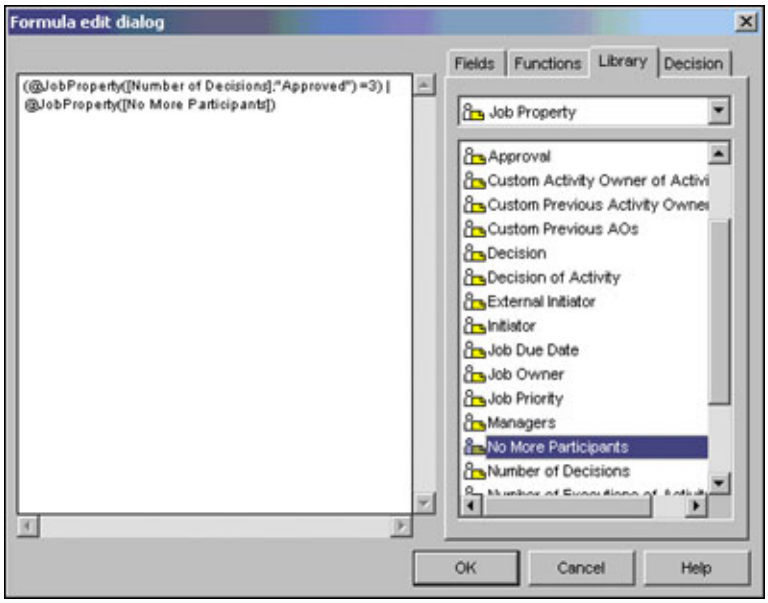
   @JobProperty([Number of Decisions];"Approved") = 3

4. To make sure we do not do the activity again if there are no more potential owners left, add the following function to the formula (again by selecting it from the JobProperties):

   @JobProperty([No More Participants])

The entire formula is:

@JobProperty([Number of Decisions];"Approved") = 3 | @JobProperty([No More Participants])



Note that in this example, the @Input parameter is Approved, but you could also specify another value for this, for instance, Rejected.

By setting the condition up in this way, it keeps routing back to the Approve n times activity until we have three

approvals or there no more people to complete the activity.

Other hints and suggestions
You might create a different loop-back process by making small changes to the formula. For example, you could also determine the number of decisions needed at runtime by asking users to fill out a field on the form. Then you would refer to the field name rather than the hard-coded number three. You could also test for a number of rejections by replacing the Decision value in the condition formula.

Also, keep in mind that this is not parallel routing. The decisions take place in sequence, but there is no defined order in which people make decisions.

Finally, the OR part of the condition formula could also be on a different routing relation. Note that this wouldn't work if you have a routing relation further down in the process that feeds back into the activity.

Summary
The two approval process customization examples we've looked at should give you an idea of the flexibility built into Workflow. Using programmatic features such as LotusScript, JavaScript, and formulas, you can customize Workflow processes in almost limitless ways.

ABOUT THE AUTHORS
Hans Boone has worked with Lotus products as a consultant since 1998. In 2000, he joined Lotus Professional Services Netherlands as a consultant working on Enterprise Integration and Workflow. He has implemented global workflow solutions for several customers.

Cees (pronounced "case") van der Woude has worked with Lotus Workflow since its inception as ProZessware, a product developed by ONEstone before the company was acquired by Lotus in 1999. His extensive and deep knowledge of Lotus Workflow has made him invaluable as a member of the Knowledge Management Enablement and Business Support Team and as a friend and advisor to the Lotus Workflow development team.

