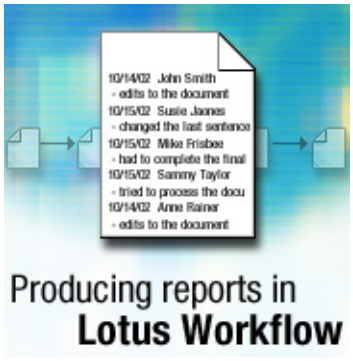
Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today



Level: Intermediate
Works with: Lotus Workflow
Updated: 02-Dec-2002

by
Cees
van der Woude

Wouldn't it be nice if you could open a Lotus Workflow document that is going through a workflow process and immediately see:

1. Where this document has been?
2. What decisions were made?
3. How long each activity had taken?

And if you stored this information inside a single binder document (rather than separate audit trail documents), you could create views that slice and dice the information in various ways to provide you and your management with valuable insights. Of course, Lotus Workflow includes viewers for presenting graphical information about single jobs. But there are times that you may prefer to see this data as text in a format that you can use to construct Workflow-related analysis reports. The following screen shows an example:

Log By Job

Job	Completed	Activity	Owner	WorkTime	h:m:s	Act
EXCR-5EDHTR				34568		
EXCR-5EDHTS				71621		
EXCR-5EDHTT				34670		
EXCR-5EDHTU				37957		
	09/27/2002 09:29:10 AM	First Mail	Excalibur	1	0:0:1	
	09/27/2002 09:42:15 AM	SUBMIT RECIPE NEED	Pieter Poors	784	0:13:4	
	09/27/2002 09:43:59 AM	Tmp Send Mail	Excalibur	104	0:1:44	
	09/27/2002 09:51:55 AM	RESEARCH	Workflow Admin	475	0:7:55	
	09/30/2002 10:03:40 AM	SUBMIT	Joe Hureau	33104	9:11:44	
	09/30/2002 11:00:19 AM	ESTIMATE	Workflow Admin	3397	0:56:37	
	09/30/2002 11:01:54 AM	APPROVE ESTIMATE	Julie Andrews	92	0:1:32	
EXCR-5EDRCC				1		
WADN-5EGQUR				2		
WADN-5EGQUS				0		
				178819		

In this article, we explain how you can create reports of this type in Lotus Workflow, using meta-data stored on the cover document. We also show you how to add custom data (such as decision comments and activity duration) to Lotus Workflow. We provide all the necessary design elements and code to do this, which you can download from

the **Sandbox**. (Note these examples only work with Lotus Workflow 3.0.1.)

This article assumes that you're an experienced Lotus Workflow programmer.

Installing the sample code

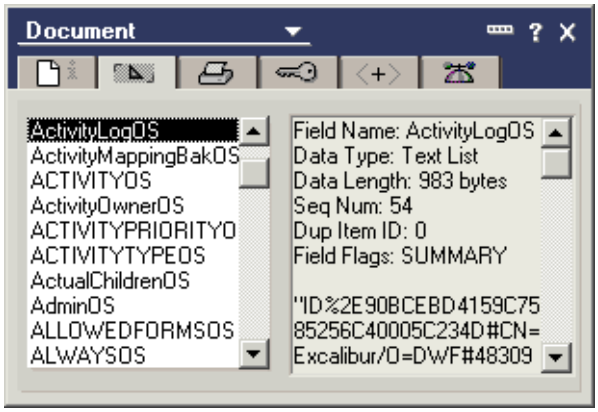
To use the samples provided with this article, do the following:

1. Download the sample database (LWFReport.nsf) from the Sandbox.
2. In Domino Designer, create a new application database based on the Lotus Workflow 3.0.1 Application Template.
3. Open LWFReport.nsf, copy its script library Time Utils, and paste it into your new application database.
4. Add a Use "Time Utils" statement to your OS Application Events Library.
5. Open LWFReport.nsf's About document. This contains two snippets of code. Copy the first and paste it into your application's OS Application Events Library in the Private function QueryActivityCompleted. Similarly copy the second sample code segment, and paste it into your OS Application Events Library under Declarations - Function QueryExecuteServerToDo.
6. Copy LWFReport.nsf's two sample views Log By Activity and Log By Job, and paste them into your application database.
7. Copy the sample subform LogTabl into your application database and insert this subform into the forms SKWebBinderCover (for Web users) and/or SK Binder Cover (for Notes users).
8. Save and close your application database.

You're now ready to use our sample code in your own application. The following sections explain what each sample component does.

Understanding ActivityLogOS

As you may know, each workflow binder consists of at least a cover document (usually not visible) and a main document (the primary interface for users). When a workflow job is moving through the process you designed in the Architect, the engine gathers information about each step. This information is stored in an internal logging field called ActivityLogOS. This field is stored only in the cover document. To inspect it, go to the Adminstration\All by Job view and find the cover document for the job. Open the Document Properties dialog box, and view the ActivityLogOS field:



For each completed activity, there is one item in the list. Each item has the following format:

ActivityID#ActivityOwner#LoopID#Unused#ActivityName#DecisionLabel#DecisionChoice#DecisionComment#CustomEntries

where:

- ActivityID is the DocUNID of the activity description document in the process definition database.
- ActivityOwner is the name of the person or server that completed the activity.
- LoopID is used internally by the Workflow engine if activity loops back onto itself.
- ActivityName is the name of the activity.
- DecisionLabel is the prompt label for the decision, as specified in the Workflow Architect.
- DecisionChoice is the decision alternative selected by the activity owner.
- DecisionComment is a comment associated with a decision.

- CustomEntries are a list of custom-defined entries separated by #.

After two activities have been processed, a typical ActivityLogOS might look like this:

"ID%F8521028A2921AC785256C1D005D8498#CN=Workflow Admin/O=DWF#6112#Submit FR####"
"ID%4312785481985119 85256C1D005D849E#CN=John Alberta/O=DWF#12515#Approve Estimate##Please decide#Rejected##"

Informative, but not particularly easy for the average user to understand. So we need a way to display this information in a clearer, more friendly way. The next section explains how.

Displaying activity information

If you want to show ActivityLogOS information to end-users in a more usable format, you can add a table to the forms (SK Binder Cover) and/or (SK Web Binder Cover). This table can display information when users click More Information\Show Detail, for example:

Activity Owner	**Team Members**
none	none
Potential Activity Owner(s)	**Additional Readers**
John Alberta/DWF; Shelly Ducklow/DWF; Loewel Goemaat/DWF; Sharon Furlong/DWF; Pieter Paars/DWF	[Process Reader]
Job Owner	**Priority**
Workflow Admin/DWF	2.Medium

Previous Activities	Owners	Previous Decisions
First Mail	Excalibur	
SUBMIT RECIPE NEED	Pieter Paars	
Tmp Send Mail	Excalibur	
RESEARCH	Workflow Admin	
SUBMIT	Joe Hureau	
ESTIMATE	Workflow Admin	
APPROVE ESTIMATE	Julie Andrews	APPROVE
CREATE OR MODIFY RECIPE	Workflow Admin	
APPROVE CONCEPT	Workflow Admin	NEED MODIFICAT

To create the table:

1. In Domino Designer, open your application database, then open the (SK Binder Cover) form.
2. Create a table consisting of two rows and three columns.
3. Add the following field in the first column:
   Name: Log
   Type: Computed for Display, multi-value, separator=New Line
   Formula: @Word(ActivityLogOS;"#";4)
4. Copy this field and paste it into the next two columns. These two fields should now appear as Log_1 and Log_2.
5. Change the formulas as follows:
   Log_1: @Name([CN];@Word(ActivityLogOS;"#";2))
   Log_2: @Word(ActivityLogOS;"#";7)
6. Save and close the form.

If you want to display this information directly from your main document, you could make the table part of the OS Domino Workflow Information subform and/or OS Domino Workflow Web subform, and change the formulas to the following format:

Log: @Word(@GetDocField(FolderIDOS;"ActivityLogOS");"#";2)

This will look up the ActivityLogOS values from the CoverDocument via its DocumentUniqueID stored in FolderIDOS.

Note: For most applications, I recommend using the table in the cover document. This way, calculations only need to be done when users select More Information\Details.

Adding decision comments

When users in a process make decisions, there's often a need to add a brief comment. It's easy to capture these comments and make them part of the ActivityLogOS. After you do this, these comments can be displayed, using the same technique described in the preceding steps.

In our example, whenever a decision is made and a comment added, the workflow engine finds a value in a field called ApprovalRemarkOS on the CoverDocument. The engine then inserts this value in the ActivityLogOS field, in the eighth position (identified in the previous section as DecisionComment). To take advantage of this, we need to make sure a comment is stored in ApprovalRemarkOS when a decision activity is completed:

1. Open your application in Domino Designer. Open the OS Domino Workflow Information subform and/or the OS Domino Workflow Web subform.
2. Create a new field just below the field ApprovalChoiceOS with the following properties:
   Name: Comment
   Input Translation formula: @SetDocField(FolderIDOS;"ApprovalRemarkOS";Comment);""
3. Copy and paste the Hide-when formula from ApprovalChoiceOS field to the Comment field. This ensures the field only appears when a decision needs to be made. The input translation formula will move whatever comment was typed in to the ApprovalRemarkOS field on the CoverDoc, and then erase the comment field. The Workflow engine updates the ActivityLogOS field and inserts the comment into DecisionComment (at position eight DecisionComment).
4. Add a column to your display table, copy the Log field again and change the formula to display position eight. If the table resides in the main document, you'll need the @Word(@GetDocField....) formula. If the table is visible on the cover, you need only the @Word(...) formula.
5. Save and close the subform.

Your display table should now appear similar to the following:

Activity Owner none			**Team Members** none
Potential Activity Owner(s) John Alberta/DWF; Shelly Ducklow/DWF; Loewel Goemaat/DWF; Sharon Furlong/DWF; Pieter Paars/DWF			**Additional Readers** [Process Reader]
Job Owner Workflow Admin/DWF			**Priority** 2.Medium
Previous Activities	Owners	Previous Decisions	Remarks
First Mail	Excalibur		
SUBMIT RECIPE NEED	Pieter Paars		
Tmp Send Mail	Excalibur		
RESEARCH	Workflow Admin		
SUBMIT	Joe Hureau		
ESTIMATE	Workflow Admin		
APPROVE ESTIMATE	Julie Andrews	APPROVE	
CREATE OR MODIFY RECIPE	Workflow Admin		
APPROVE CONCEPT	Workflow Admin	NEED MODIFICATION	Check with Joe!

Adding custom entries

As described earlier in this article, you can add your own entries into the ActivityLogOS field. You do this in the last part of the field, labeled CustomEntries (in the ninth position of ActivityLogOS). You can append multiple custom entries in this position, separating them with the # character.

In this section, we explain how to use CustomEntries to show when an activity completes and the duration of an activity.

Activity completing time

In our example, we display when an activity is completed by having the Workflow engine:

1. Copy data it finds in the CustomAlogEntryOS field in the CoverDocument
2. Add it to ActivityLogOS in the CustomEntries position

We could use the same technique described in the preceding sections: Add a field to one or both of the workflow subforms and use an input translation formula to populate the CustomAlogEntryOS field on the CoverDocument. The Workflow engine would take care of the rest. However, this raises possible issues for automated activities, in which there is no user interaction with the form (consequently the field would never be set). So instead, we use LotusScript and the Lotus Workflow events structure. For interactive activities, the event we use is QueryActivityCompleted. (Do not use any event that gets triggered later than QueryActivityCompleted because the engine will already have updated the ActivityLogOS.)

Here is our sample code to do this:


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

    CoverDocument.CustomAlogEntryOS=Cstr(Format(Now,"mm/dd/yyyy hh:mm:ss AM/PM")

End Function
```

This will work with both immediate and scheduled routing.

When considering how to handle automated activities, review the list of events in the OS Application Events Library for the appropriate event. QueryAutomatedActivityMail does not work for automated activities that run agents. QueryBackgroundAgent is not triggered for individual binders. PostFolderRouting runs too late; ActivityLogOS has already been updated, so choose an undocumented feature!

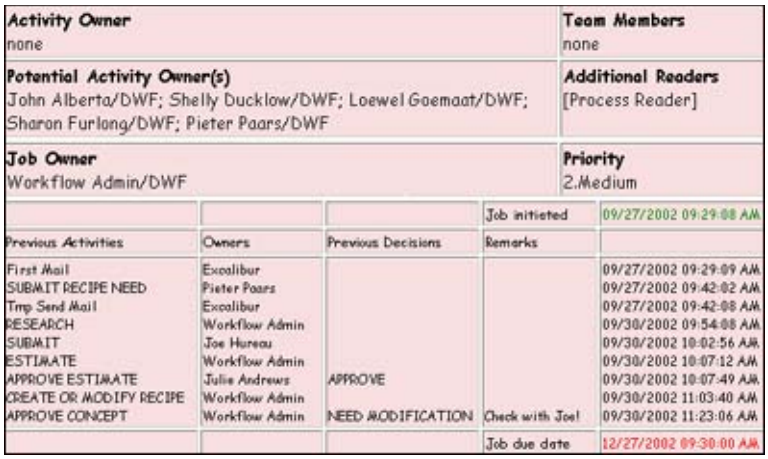
If you open your OS Application Events Library and look under Declarations, you see all the workflow event calls. One of them is:

```
Function QueryExecuteServerToDo(Continue As Integer, ToDo As Variant, ServerQueue As Variant)
```

This function call has a comment indicating it is not exposed. However, the Workflow engine triggers the event—at just the right moment for our code to set the CustomALogEntryOS field. To do this, add the following code below the QueryExecuteServerToD... function call:

```
Dim docCover As NotesDocument
' first check type of object ToDo, in case of form-based initiation it will be Nothing
If Not (ToDo Is Nothing) Then
    If Typename( ToDo ) = "NotesDocument" Then
        Set docCover = ToDo
        Else
            Set docCover = ToDo.GetCover
    End If
        docCover.CustomAlogEntryOS=Cstr(Format(Now,"mm/dd/yyyy hh:mm:ss AM/PM"))
End If
End Function
```

The time stamp can be made visible by adding a log field to our display table, and parsing out the ninth position (CustomEntries) from ActivityLogOS. (Be aware, at some stage, your table columns can become too small to display meaningful data. For example, you may want to restrict the length of decision remarks!)



Activity Owner none				**Team Members** none
Potential Activity Owner(s) John Alberta/DWF; Shelly Ducklow/DWF; Loewel Goemaat/DWF; Sharon Furlong/DWF; Pieter Paars/DWF				**Additional Readers** [Process Reader]
Job Owner Workflow Admin/DWF				**Priority** 2.Medium
			Job initiated	09/27/2002 09:29:08 AM
Previous Activities	Owners	Previous Decisions	Remarks	
First Mail	Excalibur			09/27/2002 09:29:09 AM
SUBMIT RECIPE NEED	Pieter Paars			09/27/2002 09:42:02 AM
Tmp Send Mail	Excalibur			09/27/2002 09:42:08 AM
RESEARCH	Workflow Admin			09/30/2002 09:54:08 AM
SUBMIT	Joe Hureau			09/30/2002 10:02:56 AM
ESTIMATE	Workflow Admin			09/30/2002 10:07:12 AM
APPROVE ESTIMATE	Julie Andrews	APPROVE		09/30/2002 10:07:49 AM
CREATE OR MODIFY RECIPE	Workflow Admin			09/30/2002 11:03:40 AM
APPROVE CONCEPT	Workflow Admin	NEED MODIFICATION	Check with Joe!	09/30/2002 11:23:06 AM
			Job due date	12/27/2002 09:30:00 AM

You may have to use different date/time formatting for your organization, but the basic approach we used in this example should otherwise work. Test it by running a process that has an automated activity that sends an email, runs a LotusScript agent, or uses custom code based on the Script Library class template.

Calculating, storing, and displaying activity duration

OK, so now we have the time stamp. But how long did an activity take? Let me give you an example that uses the techniques described previously. An example of what your table may look like is shown as follows:

	Team Members none			
mgat/DWF; Sharon	**Additional Readers** [Process Reader]			
	Priority 2.Medium			
		Job initiated	09/27/2002 09:29:08 AM	
	Previous Decisions	Remarks		Time
			09/27/2002 09:29:09 AM	0:0:1
			09/27/2002 09:42:02 AM	0:12:52
			09/27/2002 09:42:08 AM	0:0:6
			09/30/2002 09:54:08 AM	72:11:57
			09/30/2002 10:02:56 AM	72:8:46
			09/30/2002 10:07:12 AM	0:4:12
	APPROVE		09/30/2002 10:07:49 AM	0:0:35
			09/30/2002 11:03:48 AM	0:55:49
	NEED MODIFICATION	Check with Joe!	09/30/2002 11:23:06 AM	0:19:21
		Job due date	12/27/2002 09:30:00 AM	

First, let's add the necessary code to the QueryActivityCompleted event:

```
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 Tsec As Long
    Dim Current As NotesDateTime
    Dim item As Notesitem
    Dim Inboxdate As NotesDateTime
    Dim ActualTime as String
    Set current = New NotesDateTime(Now)
    ' if this is the first activity there is no IndateOS so we'll use JobStartedOS
    If coverdocument.HasItem("IndateOS") Then
        Set item=coverdocument.GetFirstItem("IndateOS")
    Else
        Set item=coverdocument.GetFirstItem("JobStartedOS")
    End If

    Set inboxdate=item.DateTimeValue
    ' Calculate the difference between Now and indate or startdate
    Tsec = current.TimeDifference(inboxdate)
    ' Make sure OS Application Events library contains Use "Time Utils" to call GetExcelTime function
    ActualTime=GetExcelTime(Tsec)
    ' Append the number of seconds as a text string to the CustomAlogEntryOS, separate from timestamp with a #
    CoverDocument.CustomAlogEntryOS=Cstr(Format(Now,"mm/dd/yyyy hh:mm:ss AM/PM") & "#" & ActualTime)

End Function
```

In the preceding code, Tsec calculates the number of seconds that have passed between the moment the activity arrived (InDateOS) and the moment of completion (@Now). If this is the first activity in the process, there will be no IndateOS, so we use the moment the job was started (JobStartedOS). The number of seconds is then passed to a function GetExcelTime (which we'll discuss further later in this article) to transform into a string in hh:mm:ss format. This string then gets stored in CustomAlogEntryOS. Note the # separator, which causes the duration value to become the tenth word in ActivityLogOS.

To accommodate automated activities, we add code to the QueryExecuteServerToDo event. However, there's a glitch: When the Workflow engine moves a binder to an automated activity, it does not update the IndateOS field. So instead of looking for InDateOS, we extract the completion date from ActivityLogOS. Again, if this is the first activity, we use JobStartedOS:


```
Function QueryExecuteServerToDo(Continue As Integer, ToDo As Variant, ServerQueue As Variant)
    Dim docCover As NotesDocument
    Dim Tsec As Long
    Dim Current As NotesDateTime
    Dim item As Notesitem
    Dim datestring As Variant
    Dim ActualTime as String
    Dim Inboxdate As NotesDateTime
    If Not (ToDo Is Nothing) Then

        If Typename( ToDo ) = "NotesDocument" Then
            Set docCover = ToDo
        Else
            Set docCover = ToDo.GetCover
    End If

     If docCover.FolderStatusOS(0)="Automation" Then
     Set current = New NotesDateTime(Now)
            ' if there is no ActivityLogOS yet, then this activity is the first one and we should take JobStartedOS
            If docCover.Hasitem("ActivityLogOS") Then
                ' get the previous completion timestamp and treat that as the Indate
                datestring=Evaluate(|@Word(@SubSet(ActivityLogOS;-1);"#";9)|,docCover)
            Else
                datestring=Evaluate(|JobStartedOS|,docCover)
            End If
            Set inboxdate=New NotesDateTime(datestring(0))
            Tsec = Cstr(current.TimeDifference(inboxdate))
            ' Make sure OS Application Events library contains Use "Time Utils" to call GetExcelTime function
            ActualTime=GetExcelTime(Tsec)
            ' Append the number of seconds as a text string to the CustomAlogEntryOS, separate from timestamp
            with a #
            CoverDocument.CustomAlogEntryOS=Cstr(Format(Now,"mm/dd/yyyy hh:mm:ss AM/PM") & "#" &
            ActualTime)
        End If
    End If
End Function
```

You can now record the activity information you want. But before displaying this information in views, you may want to perform one more step to ensure time data is calculated correctly: Make sure non-office hours, such as nights and weekends, are not factored in. We explain how to do this in the next section.

Removing non-office hours from time calculations

Imagine you route an activity to someone at 5 pm on Friday. You're still working, but the person to whom you've routed the activity has gone home. When that person comes in Monday morning at 8:30, the first thing he does is complete the activity. In "calendar time," it's taken over 60 hours to complete the activity. But in reality, it may have required no more than a few working minutes. So to accurately capture the real duration of an activity, it's best to include only time the assigned person is actually at work.

Luis Veloso has written code to do this and posted it in the **Sandbox**. This code returns the difference between a starting date/time and an ending date/time, excluding non-office hours and weekend days. To use this code, download it from the Sandbox. Then open your application in Domino Designer, and open the Time Utils script library. Open the Options section and change the StartHDay and EndHDay constants. Then paste in the following code from the sample into QueryActivityCompleted. (In the code below, the lines changed by the sample appear in bold.)

```
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 Tsec As Long
    Dim Current As NotesDateTime
    Dim item As Notesitem
    Dim Inboxdate As NotesDateTime
```



```
    Dim worktime As String
    actualtime As String
    Set current = New NotesDateTime(Now)

    If coverdocument.HasItem("IndateOS") Then
        Set item=coverdocument.GetFirstItem("IndateOS")
    Else
        Set item=coverdocument.GetFirstItem("JobStartedOS")
    End If

    Set inboxdate=item.DateTimeValue
    Tsec = current.TimeDifference(inboxdate)
    actualtime=GetExcelTime(Tsec)
    worktime=workhours(inboxdate,current)

    CoverDocument.CustomAlogEntryOS=Cstr(Format(Now,"mm/dd/yyyy hh:mm:ss AM/PM") & "#" &
    actualtime & "#" & worktime)

End Function
```

and QueryExecuteServerToDo:

```
    Function QueryExecuteServerToDo(Continue As Integer, ToDo As Variant, ServerQueue As Variant)
        Dim docCover As NotesDocument
        Dim Tsec As Long
        Dim Current As NotesDateTime
        Dim item As Notesitem
        Dim datestring As Variant
        Dim Inboxdate As NotesDateTime
        Dim worktime As String
        Dim actualtime As String
        If Not (ToDo Is Nothing) Then

            If Typename( ToDo ) = "NotesDocument" Then
                Set docCover = ToDo
            Else
                Set docCover = ToDo.GetCover
            End If

            If docCover.FolderStatusOS(0)="Automation" Then
                Set current = New NotesDateTime(Now)
                If docCover.Hasitem("ActivityLogOS") Then
                    datestring=Evaluate(|@Word(@SubSet(ActivityLogOS;-1);"#";9)|,docCover)
                Else
                    datestring=Evaluate(|JobStartedOS|,docCover)
                End If
                Set inboxdate=New NotesDateTime(datestring(0))
                Tsec = Cstr(current.TimeDifference(inboxdate))
                actualtime=GetExcelTime(Tsec)
                worktime=workhours(inboxdate,current)
                docCover.CustomAlogEntryOS=Cstr(Format(Now,"mm/dd/yyyy hh:mm:ss AM/PM")&"#"&
                actualtime & "#" & worktime)
            End If
        End If
    End Function
```

Note the following:

- If you use scheduled routing, the time between completing the activity and the arrival at the next activity is not included. Consider any discrepancy between the sum of all Actual times and the difference (JobCompletedOS - StartDateOS) to be "routing time."
- When overtime-working employees complete activities outside regular office hours, ActualTime is displayed as 0:0:0.

Creating activity views

Now that we've captured all this activity information, we can easily create views to display it. For example, you can create a view that displays ActivityLogOS information by activity:

New request | My activities / Jobs I started | All jobs / Jobs by status | By ingredient | Log by activity / Log by job

Log By Activity

Activity\Job	Completed	Owner	ActualTime	h:m:s
APPROVE CONCEPT			1161	
APPROVE ESTIMATE			127	
CREATE OR MODIFY RECIPE			3349	
ESTIMATE			3649	
First Mail			6	
RESEARCH			521108	
EXCR-5EDHTU	09/27/2002 09:51:55 AM	Workflow Admin	475	0:7:55
EXCR-5EDHTS	09/30/2002 09:54:08 AM	Workflow Admin	259917	72:11:57
EXCR-5EDHTR	09/30/2002 10:05:17 AM	Julie Andrews	260716	72:25:16
SUBMIT			519629	
SUBMIT RECIPE NEED			263490	
Tmp Send Mail			295	
			1312814	

This view is included in our sample database, LWFReport.nsf. To use it in your application database, simply open Domino Designer, copy the Log by Activity view from LWFReport.nsf, and paste it into your application.

The selection formula for Log by Activity is as follows:

```
SELECT CoverDocOS="yes" & ActivityLogOS!=""
```

The following tables lists columns for this view:

Name	Formula	Cate-gorize?	Show multiple values as separate entries?	Totals
	@Word(ActivityLogOS;"#";4)	Y	Y	
Activity/Job	InstanceOS	N	N	
Completed	@Word(ActivityLogOS;"#";9)	N	Y	
Owner	@Name([CN];@Word(ActivityLogOS;"#";2))	N	Y	
Time (seconds)	ExcelTime:=@Word(ActivityLogOS;"#";10); hr:=@TextToNumber(@Word(ExcelTime;":";1))*3600; min:=@TextToNumber(@Word(ExcelTime;":";2))*60; sec:=@TextToNumber(@Word(ExcelTime;":";3)); hr+min+sec	N	Y	Y
h:m:s	@Word(ActivityLogOS;"#";10)	N	Y	

The Log by Activity view shows jobs categorized by activity. It lists completion dates, activity owners, durations, and totals. Our sample also includes a Log by Job view. You can also create your own views, for example By Owner.

Before we end this section, we'd like to leave you with a few points to consider:

- With large processes, the ActivityLogOS may overflow. Lotus Workflow provides built-in support to continue the log in the file ActivityLogOS_1. In such cases, you must take this into account in your tables and view column formulas.
- When using Notes, make sure client clocks are synchronized with server's clock, and system date/time

settings are the same.
- Unlike the Lotus Workflow audit trail, the technique described here does not create additional documents in the database. However, the view indexes for the Log views can grow extremely large because of the multi-value settings. Consider using manual refresh on these views. You can also discard these view indexes after a specified time of inactivity. (As a general practice, we recommend archiving completed jobs as soon as possible. For historical analysis, you can use the Log Views in archive databases.)

Lotus Workflow reporting: good news

As we've demonstrated in this article, it isn't particularly difficult to implement activity reporting in Lotus Workflow, especially if you use the samples we provide. You can present report information in several different ways, including adding custom data. You can also create views to display this information in a way most meaningful to your users.

We hope you've found this article useful. For more information about Lotus Workflow, see its **product page** on LDD. Also consult the **Lotus Workflow documentation**, which you can download from the **Documentation Library**.

ABOUT THE AUTHOR
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.

