

## Notes 6.x Application Strategies: Document Rating

We are sure that you can recite the same clichés about the importance of knowledge that we can:

- *Garbage in, garbage out.*
- *Data → Information → Knowledge.*
- *Knowledge is power.*

Notes makes it easy to create information, and there is a concerted effort to make it easy to locate specific information, all in the hope of turning all that information into knowledge. The full-text search engine plays a vital part in this effort.

We suggested in the article [Notes 6.x Application Strategies: Interactive Search](#) that you can more fully exploit the full-text search engine by adding a custom search form to your applications. This helps find information in a single database, but to find it across databases, you will likely employ these other IBM Lotus search innovations (that leverage the full-text search engine):

- [Domain Search](#) is built into Domino. It lets you simultaneously full-text search multiple Notes databases and file system resources (anything from word processing to HTML files).
- [Lotus Extended Search](#) extends simultaneous full-text search to also include RDBMS, Index and Directory sources, Content Management applications, Lotus Instant Messaging and Web Conferencing (Sametime) users, and more.
- The [Lotus Discovery Server](#) goes further than just searching by automatically ranking and associating information from many repositories (documents, expertise profiles, Lotus Team Workplace (QuickPlace) places, relational databases, Web pages, and so on) according to such things as:
  - Information usage (number of links to it and frequency of use)
  - Relationship of the information to a pre-defined knowledge map (a list of categories)
  - People's expertise strength and affinity to particular topics
  - The value of a piece of information relative to other documents in a category (in the knowledge map) across repositories

The promise of Discovery Server is to increase the relevance of searches by showing how each piece of datum relates to other information.

All this great technology keeps us from getting lost in the information forest. The *ability* to find relevant and related information does not necessarily mean, however, that you *will* find sound, reliable, or helpful information. So let's step back to get some perspective and ask a fundamental question about search technology:

*Is it even possible for a computer to determine the worth of information?*

We would venture to say that at this point in the development of information technology the answer is still "no." Only people can determine the worth of information. And because Notes is the champ of collaborative tools, it makes sense that Notes databases are ideal not only for storing information in the first place, but also for providing the forum for a community of users to judge the worth of that information. That communal judgment then becomes a vital aspect of the information.

In this article, we show you how to easily add a subform to your forms to rate the worth of documents, an agent that tallies the ratings, and views that sort documents by ratings.

To combine technology that finds information with the human rating of its worth is perhaps our best chance at turning data into knowledge.

---

## Install the example database

The code shown in this article is from the Document Rating database (`documentrating.nsf`), which you can download from the Sandbox.

To get the application ready for demonstration, your server administrator must perform these steps:

- Copy the database to the data directory on a non-production server.
- Assign you Designer-level or greater rights and assign the [DBAdmin] role to you.
- Assign the server (and the group of servers) Manager access and assign the entry to the [DBAdmin] role.
- Assign the server administrator (or the group of administrators) Manager access.
- Sign the database elements with a Notes User ID from your Organization. To do this, open the Files function tab in Domino Administrator. Right-click the database. Choose **Sign**. Use the Active Server's ID to sign all the design elements.

---

## Configure the application

Before we can see how document rating works, the database administrator must first configure the application.

Let's open and configure the database to enable document rating. It is assumed that the

database has been installed on the server as described above.

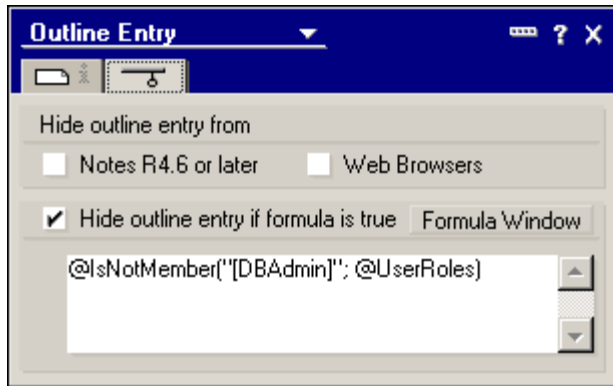
1. Open the database using Notes 6.x.
2. If you get a version of the database with the profile document deleted, you will be taken immediately to the Application Settings form:

The screenshot shows the 'Application Settings' form for 'Document Rating'. At the top, there is a 'Save & Close' button with a green checkmark. Below this, the 'Application Settings' title is displayed in a blue box. To the right, the 'Last Modifier' is 'Kent Kurchak/wareSource' and the 'Last Modified On' field is empty. The 'Application Title' is 'Document Rating'. Below this, there are two tabs: 'Database Open' and 'Document Rating'. The 'Document Rating' tab is active, showing 'Active Users: 1'. There is a button labeled 'DeActivate DailyUserActivity Agent'. To the right of this button, there is a text box explaining the agent: 'The DailyUserActivity agent runs daily and is responsible for collecting user activity that occurs over multiple server or local replicas into a single User Activity document. This agent cleans up the replication conflicts seen in the User Activity view.'

If the application has already been configured, the application opens to a page with an embedded outline control in the left frame and the FAQs view in the right frame:

The screenshot shows the application interface. On the left, there is a workspace with a folder icon and the text 'Workspace'. Below this, there is a tree view with the following items: 'Application Settings (Profile Doc)' (expanded), 'Delete ProfileDoc (For testing only!)', 'FAQs (View)', 'Admin Views' (expanded), 'Rating Queue', and 'User Activity'. On the right, there is a 'New FAQ' button and a 'FAQ ^' dropdown menu. The dropdown menu is open, showing three items: 'Application Development', 'LotusScript', and 'Troubleshooting'.

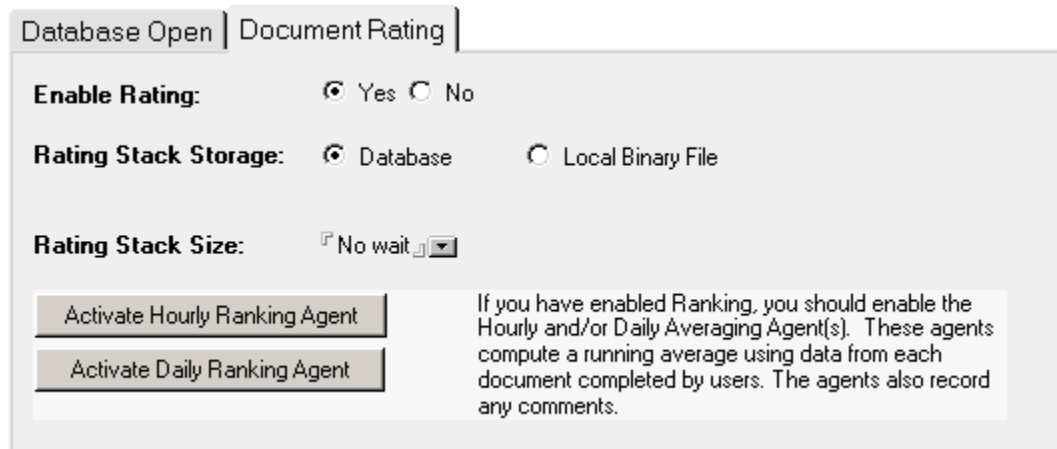
By the way, the Outline control entries all have Hide When formulas. For example, who can edit the profile document is limited to one or more database administrators, defined in the ACL as members of the [DBAdmin] role:



Because you are in the [DBAdmin] role, you can also see the administration views.

Click the **Application Settings (Profile Doc)** entry to see the configuration settings.

3. Click the Document Rating tab and change the Enable Rating option to **Yes**:



Leave all the other settings at their default values. (They will be defined later in the article.)

4. Click the **Save & Close** action button to save the profile document with the default application settings.

---

## Enable a document to be rated

Now that document rating is enabled for the application, it is up to the document authors (or someone with Editor+ access) to enable rating for each document.

In the example database, rating has not been enabled for any of the documents. Follow these steps to enable document rating for one of the documents:

1. Click the FAQs entry in the outline to open the FAQs view.
2. Expand one of the categories and open any document.

- Double-click the document or press Ctrl+E to switch the document to Edit mode.
- Scroll down to the bottom of the document to the Rate? radio button:

**Rate?**  No  Yes

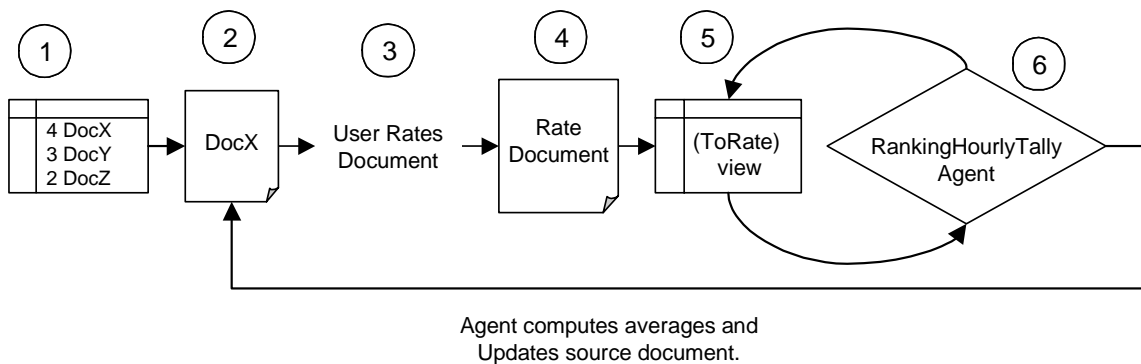
- Select **Yes** to enable rating for this document. Now choose which type of rating you want users to perform:

**Rate?**  No  Yes  
**Type:**  Importance  Interest  Relevance  Usefulness

- Save and close the document.

## Rate a document

Before explaining the code behind the Document Rating application, let's look at a diagram of the overall process of how documents are rated:



Here are the steps in the flow of the application:

- The user locates a document from a view that sorts the documents by rating.

FAQ ^
▼ <b>Application Development</b>
⑤ Using the object-oriented features of LotusScript
④ LotusScript: Rich text objects in Notes/Domino 6
③ LotusScript: Programming views in Notes/Domino 6
▼ <b>LotusScript</b>

- The user opens the document for reading.

- While reading the document, the user scrolls to the bottom of the document to see the current rating data:

**Please take a moment to [rate](#) this document:**

**Importance Rating**

Respondents:	2					Average:	1.50
Rating:	1 <-----> 5	Not Important	Not Very Important	Somewhat Important	Very Important	Extremely Important	
Totals:		1	1				

The type of rating (Importance Rating in this case), number of respondents, average, and tally of responses under each response is displayed. If anyone entered a comment, it appears below the ratings.

**Note:** It is an added bonus that when the full-text index is updated, the comments are also included and now available for searching. This adds to the richness of the results returned by a full-text search, perhaps leading to serendipitous discoveries.

The user clicks the [rate](#) link and a dialog box opens, presenting the user with a radio button to select a rating and a comment field.

**Ranking By Importance**

**Importance Rating**

Please rate this document using the following scale:

Extremely Important

Very Important

Somewhat Important

Not Very Important

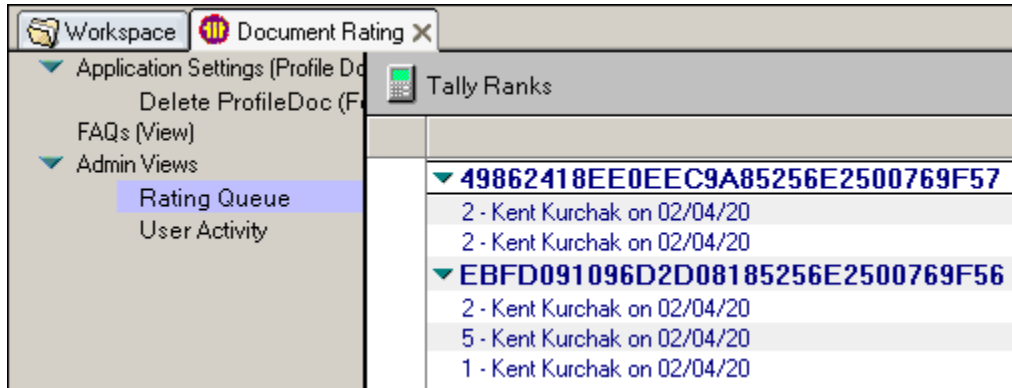
Not Important

**Comments:** I especially liked the treatment of XML with respect to rich text fields. Thanks!

Thank you for rating this document! Your rank will be tallied at the end of the hour.

OK

- When the user clicks OK to close the dialog box, the results are saved to a hidden Rate document.
- Rate documents only appear in the hidden ToRate view. Click Rating Queue in the outline to open the DBAdmin-only view of Rating documents that need to be processed:



Expand one of the categories (rated document UNID) to see the Rating documents.

- The hourly RankingHourlyTally agent processes the Rate documents, updating the rating data of the corresponding document. (A duplicate agent RankingDailyTally agent can be enabled along with/instead of the hourly version.) The Rate document is deleted after processing so that the ratings are anonymous. Comments are not anonymous because the user name and date are displayed. (If you don't want to wait for the agents to run on schedule, click the Tally Ranks button to immediately process the Rate documents.)

---

## Rating and document security

The architectural challenge of this application is updating documents that users don't have the right to edit. How is this accomplished?

First off, it *is* required that users have Author (and Create documents) access to the database, as they need to create the Rating documents, albeit using back-end LotusScript methods. This does not mean, though, that users necessarily have the right to create any type of other documents in the database. You can control who can create documents with forms on the Security tab of the Form Properties box.

Nor do users need any type of effective editor access (afforded using an Authors field) to any documents to modify their rating tally. This is because the RankingHourlyTally and/or the RankingDailyTally agents are responsible for updating documents on behalf of the users who do not have sufficient access.

---

## Tally agents

So let's look at the RankingHourlyTally and RankingDailyTally agents and the Tally Ranks button in the (To Rate) view that are responsible for tallying the ratings and updating documents.

The agents and the button each run the same RankAgent sub defined in the Ranking LotusScript Library.

The RankAgent sub performs these tasks at these line numbers:

- **3 - 12** Declare various variables and instantiate various objects.
- **13 - 14** Instantiate an empty NotesDocumentCollection named deletecollection that will be used to keep track of Rate documents that have been processed.
- **16 - 28** Define two rich text styles for the comments fields (one for the name/date and the other for the comment):

Kent Kurchak/wareSource on 2/2/2004 5:13:15 PM  
I especially like the treatment of XML with respect to rich text fields. Thanks!

- **31 - 32** Set up a loop through the Rate documents in the ToRate view.
- **35 - 36** Instantiate a NotesDocument object representing the content document (in this case an FAQ) and if it exists, keep going. If it can't be found, skip over the Rate document processing.
- **39 - 71** Using the number from the Rate document, perform the math and set the rating fields in the content document.
- **73 - 101** If there are any comments in the Rate document, prepend them to the other comments in the content document.
- **103** Save the content document, as no more changes need to be made to it.
- **106** Add the Rate document to the deletecollection collection. Why not just delete the Rate document? Because when you're in a loop through a view, you can't delete the current document object, as it is needed by the GetNextDocument method to find the next document.
- **111 - 114** Remove (delete) all the Rate documents that were added to the deletecollection NotesDocumentCollection.

---

## Activate the tally agents

You saw earlier that the Tally Ranks button manually runs the RankAgent sub. Obviously, it makes more sense to enable one or both of the agents that run the same code on an hourly and daily schedule. To enable the agents, click Application Settings entry in the outline control, click the Document Rating tab, and click the Activate buttons to enable the agent(s):

Database Open | Document Rating

**Enable Rating:**  Yes  No

**Rating Stack Storage:**  Database  Local Binary File

**Rating Stack Size:**

If you have enabled Ranking, you should enable the Hourly and/or Daily Averaging Agent(s). These agents compute a running average using data from each document completed by users. The agents also record any comments.

When you activate either agent, you are prompted for the server name on which the agent should run. The agent is activated and the button changes to allow deactivation.

The daily agent runs at 3:15 AM, but you can change this from the Agent List.

---

## Preventing repeated ratings

In the introduction above, we made this claim:

*The ability to find relevant and related information does not necessarily mean, however, that you will find sound, reliable, or helpful information.*

Obviously the freedom of a user to rate a document repeatedly challenges the credibility of the entire rating system.

So to prevent repeated ratings by the same user of the same document, you can set the size of the rating stack, a list of UNIDs of documents that the user has rated. Each user's rating stack is managed in a FIFO order.

To set the rating stack size, click the Application Settings entry in the outline control, click the Document Rating tab, and change the Rating Stack Size value to the desired number:



In this example, the user must rate 25 other documents before a UNID is pushed off the rating stack. If the user attempts to rate a document that has its UNID in the rating stack, this message is displayed:



All this happens when the user clicks the Rate link at the bottom of a content document. We'll show you this code later.

We decided not to store the rating stacks in the content documents for obvious reasons (document security and high potential for replication conflicts). Instead, you can choose to store the rating stack either in the database itself or in a local file.

---

## Rating Stack Storage: Database

Before talking about where we keep the rating stack in the database, we have to talk about how the database automatically tracks users opening the database.

**Note:** This aspect of the application is borrowed from the article [Notes application strategies: User Activity Tracking](#), where you can learn more about the code that is behind database activity tracking.

To see how activity tracking works, click User Activity under the Admin Views. What's this? There is a document with your name on it! Open the document to display the information recorded when you opened the database:

User Activity	Kent Kurchak/wareSource
<b>First Visit:</b>	02/03/2004 10:17 AM
<b>Most Recent Visit:</b>	02/03/2004 11:27 AM on PRO/wareSource
<b>Total Visits:</b>	5
<b>Notes Version:</b>	Release 6.5 September 26, 2003
<b>Notes Build Version:</b>	194
<b>Platform:</b>	Windows/32
<b>Rated Documents:</b>	49862418EE0EEC9A85256E2500769F57 EBFD091096D2D08185256E2500769F56 7F906602718F00E685256E2500769F58

Your first and last visits, total number of visits, and information about your version of Notes is recorded.

The nice thing about already having a UserActivity document for every user means that you now have an application-generated place to store all sorts of user-specific data. And that is exactly where you will store the list of UNIDs of documents the user has rated. In the above example, the user has rated three documents.

We decided for no particular reason that the top limit of the rating stack should be 50. You can increase this number. It seems like the multi-value field that holds the UNIDs can store something like 900 or so entries, but we kept it capped at 50.

**Note:** If you are testing this application using this storage option and want to clear your own rating stack, just delete your UserActivity document. We also added a Purge Ratings button that displays when you read an FAQ. (It is defined in the Rate subform and is hidden from non-DBAdmin users.) You can use this action to clear all the rating-related items from the document.

---

## Rating Stack Storage: Local File

If you decide not to track users who open the database, you can instead store the rating stack as a file in the user's local Notes executable directory.

The advantage of using a local file over UserActivity documents is that the Notes database will be smaller (and you avoid privacy concerns). The disadvantages are:

- The code signer must be given Execution Control List rights to the file system.
- If users switch computers, the local file is machine-specific and doesn't travel with the user. If users share a computer, the local file is named using the database name (but you can change this to use the database + user name instead).

The file is opened as a random file (made up of a series of records of identical length). We use the simple commands to write (Put), read (Get), and find (Seek) data in the file.

**Note:** It would be a simple matter to change the location of the file to somewhere other than the executable directory.

## Rate link code

The code for the Rate link is found in the Rate subform. This subform is added to every content document form (in this example, the InfoDoc form) as a computed subform using this formula:

```
@If(@GetProfileField("ProfileDoc"; "EnableRanking")="1"; "Rate"; "")
```

As such, the profile document setting to enable rating determines if the subform is embedded at run time.

The Rate subform itself is split into two halves by opposite Hide When formulas: what document authors see and what document readers see:

RatingNames T					
Show for Readers					
Please take a moment to <a href="#">rate</a> this document:					
DisplayRankType T Rating					
Respondents:	ResponseCount#	Average:		Average #	
Rating: 1 <-----> 5	Rating1NameT	Rating2NameT	Rating3NameT	Rating4NameT	Rating5NameT
Totals:	Rating1Total#	Rating2Total#	Rating3Total#	Rating4Total#	Rating5Total#
RankingComments T					
Show for Authors					
Rate?	<input type="radio"/> RankChoice				
Type:	<input type="radio"/> RankType				

The code for the Rate link is fairly simple as long as you are aware of the interplay between the execution of the Rate link code and the code that runs when we launch another subform in a dialog box. The hotspot code finishes and control of the application is transferred over to the subform in the dialog box.

To ensure a successful handoff, we set an environment (Notes.ini) variable to the UNID of the current document from the Rate link (which will be picked up by the subform when the Rate document is saved). We do this in the following line in the Rate link Click event:

```
12 Call s.SetEnvironmentVar( "SHDDocID", doc.UniversalID)
```

Before we get to the part of the code where the user rates the document, we want to make sure that the current document UNID is not in the rating stack. Remember that you can choose to store the rating stack in the UserActivity document or in a local file, so there are two cases:

```
14 Select Case profiledoc.DBOrFile(0)
15   Case "1" 'store UNIDs in database UserActivity doc
16     If FindID(s, db, doc, profiledoc) = True Then
17       MsgBox msg, 64, "Notice"
18       Exit Sub
19     End If
20   Case "0" 'store UNIDs in random file
21     If FindIDinFile(s, db, doc, profiledoc) = True Then
22       MsgBox msg, 64, "Notice"
23       Exit Sub
24     End If
25 End Select
```

The two functions, FindID and FindIDinFile (from the Ranking LotusScript Library) do all the work here in terms of adding the current document UNID to the rating stack. If the UNID is listed in the respective rating stacks, the functions return True.

If the chosen function returns False, the code continues. Regardless, the current document UNID has been added to the rating stack.

Now we can get to the business of prompting the user to rate the document. The first order of business is to create a new NotesDocument object to pass to the DialogBox method:

```
28 Dim NewDoc As NotesDocument
29 Set NewDoc = db.CreateDocument
```

This is really just a fake document to which we can associate the dialog box that opens. Normally when the DialogBox method is used, the values gathered are written back to the underlying document. But in this case, we don't want any values from the dialog box to be

saved back to the content document (especially because the user doesn't have rights to make edits). So we create a fake/temporary document instead.

Keep in mind that there are four subforms, one for each type of rating (importance, relevance, usefulness, interest). We could have used one subform to handle all four and done some conditional display magic, but it just seemed easier to keep the four types separate.

Using the type of rating selected by the document author, we select the appropriate subform to display using the DialogBox method:

```
30 Select Case doc.RankType(0)
31 Case "1" ' Importance
32 DialogFlag = w.DialogBox( "RankingbyImportanceDialog",True, True,
    True, True , True,False , "Ranking By Importance", NewDoc, True)
33 Case "2" ' Relevance
34 DialogFlag = w.DialogBox( "RankingbyRelevanceDialog",True, True,
    True, True , True,False , "Ranking By Relevance", NewDoc, True)
35 Case "3" ' Usefulness
36 DialogFlag = w.DialogBox( "RankingbyUsefulnessDialog",True, True,
    True, True ,True, False , "Ranking By Usefulness", NewDoc, True)
37 Case "4" ' Interest
38 DialogFlag = w.DialogBox( "RankingbyInterestDialog",True, True,
    True, True , True, False , "Ranking By Interest", NewDoc, True)
39 End Select
```

---

## FindID function

The FindID function is called if the rated document UNID is stored in the user's UserActivity document.

The FindID function performs these tasks at these line numbers:

- **3** Read the current document's UNID and assign it to the ThisID variable.
- **5** Find the user's UserActivity document using the GetActivityDoc function called from the UserActivity LotusScript Library. If there is no UserActivity document, then create one.
- **7 - 17** Read the profile doc to determine the rating stack size. If set to "0" (don't wait), then bypass the rest of the code and return False, meaning that the user can rate the document.

- **24** Using the `Arraygetindex` function, see if the current document's UNID is in the rating stack read from the `UserActivity` document. If it is in the list, return `True` and exit the function, meaning that the user has already rated the document.
- **30** Because the current document UNID is not in the rating stack, append the UNID to the bottom of the array of UNIDs.
- **33 - 40** Resize the array of UNIDs to match the rating stack size by copying that number of elements from the bottom of the array to a new array.
- **43 - 44** Copy the new array (now the size of the rating stack) to the `UserActivity` document and save the document.
- **45** Return `False` to the function variable, meaning that the user can rate the document.

---

### **FindIDinFile function**

The `FindIDinFile` function is called if the rated document UNID is stored in a local file.

The `FindIDinFile` function performs these tasks at these line numbers:

- **4** Read the current document's UNID and assign it to the `ThisID` variable.
- **6 - 16** Read the profile doc to determine the rating stack size. If set to "0" (don't wait), then bypass the rest of the code and return `False`, meaning that the user can rate the document.
- **18 - 19** Determine a name for the random file using the database name.
- **16 - 17** Open the random file for I/O operations.
- **26 - 34** Loop through the top records (equal to the rating stack size) in the file to see if the current document's UNID is in the rating stack. If it is found, return `True` and exit the function, meaning that the user has already rated the document.
- **38 - 41** Because the current document UNID is not in the rating stack, append the UNID to the top of file and delete the last record. Close the file, which saves it.
- **44** Return `False` to the function variable, meaning that the user can rate the document.

---

### **Rating subform Queryclose event**

Let's return to the Rate link code. Remember that the last thing it did was to launch one of the four rating subforms in a dialog box. As part of that code, we saved the current document UNID to the environment.

At this point in the code execution, the dialog box has been launched and the code from Rate link is no longer in control. The user is now looking at one of the rating subforms open in a dialog box, for example:

Ranking By Importance	
<b>Importance Rating</b>	<b>Please rate this document using the following scale:</b> <input type="radio"/> Extremely Important <input type="radio"/> Very Important <input type="radio"/> Somewhat Important <input type="radio"/> Not Very Important <input type="radio"/> Not Important
<b>Comments:</b>	I especially liked the treatment of XML with respect to rich text fields. Thanks!
	Thank you for rating this document! Your rank will be tallied at the end of the hour.

What happens next? The user fills out the form and clicks OK, and then the subform Queryclose event runs this sub (the subform includes the Ranking LotusScript Library):

```
Call SaveRankDoc()
```

The job of the SaveRankDoc sub is to create a new Rate document and record the user rating and comment along with the UNID that it picks up from the Notes.ini variable set by the Rate link.

The SaveRankDoc sub performs these tasks at these line numbers:

- **11** Instantiate a new NotesDocument object representing the Rate document that will be saved by the sub.
- **15** Read the variable from the environment (contains the rated document UNID).
- **17** Using the UNID, open the content document to read some of its items.
- **20 - 36** Set the items in the Rate document, using values from the content document and from the current document object (the subform opens in a dialog box). Also set two Authors and one Readers fields to secure the information.
- **37** Save the Rate document.

At this point, the dialog box closes and the user focus is returned to the underlying content document. If the user clicks the Rate link again, the whole process starts over. But because the user just rated the document, the UNID was just recorded and the user's attempt to rate the same document again is rebuffed.

---

## UserVisits conflicts

One important aspect of the application that was borrowed from the article [Notes application strategies: User Activity Tracking](#) is how UserActivity replication conflicts are handled.

Replication conflicts occur when a user has opened different replica copies of the database on different servers or even a local replica. Each replica of the application records and updates its own UserActivity documents. Following replication, the impact to the UserVisits view is unfortunate, but unavoidable given the distributed nature of the Notes/Domino environment:

Merge Visits					
User ▾	Visitor	First Visit	Last Visit ▾	Total Visits	
Bob Barker/wareSource	1	01/25/2004	01/26/2004	5	
Kent Kurchak/wareSource	2	01/27/2004	01/27/2004	8	
Kent Kurchak/wareSource	3	01/27/2004	01/27/2004	1	
◆	[Replication or Save Conflict]				
◆	[Replication or Save Conflict]				
					21

Duplicate documents and replication conflict documents abound.

There are three ways to resolve this problem:

- Manually delete the duplicate documents. This is the least attractive option!
- Enable the DailyUserActivity agent, which calls a cleanup routine to merge conflict documents and retain the earliest first visit time/date and the latest last visit time/date. This routine, ProcessActivityView, is also in the UserActivity LotusScript Library. As the routine iterates the NotesViewEntryCollection of the UserVisits view, it calls the MergeConflicts sub (also in the UserActivity LotusScript Library) when it finds that the user name of the current ViewEntry is the same as the next one.

MergeConflicts is actually responsible for meshing the two documents into one so that the two time/date values are the earliest and latest and the number of total visits is the higher of the two. When the agent finishes, the duplicates are removed and order is restored to the UserVisits view:

Merge Visits					
User ▾	Visitor	First Visit	Last Visit ▾	Total Visits	
Bob Barker/wareSource	1	01/25/2004	01/26/2004	5	
Kent Kurchak/wareSource	2	01/27/2004	01/27/2004	8	
					13

You can, of course, change the schedule of the agent from the Agent list so that it runs more or less frequently, depending on your needs.

- Open the UserVisits view and click the Merge Visits button. This calls the same routine that the DailyUserActivity agent calls. We use this button when testing the application across replica databases.

The last part of the MergeConflicts sub copies all of the UNIDs of rated documents from the conflict document and appends them to the list in the main document.

```
60 If parentDoc.HasItem("RankedDocs") Then
61   Set item = parentDoc.GetFirstItem("RankedDocs")
62   Forall x In entryDoc.RankedDocs
63     Call item.AppendToTextList(x)
64   End Forall
65   parentDoc.RankedDocs = Evaluate(|@Sort(@Unique(@Trim(RankedDocs)))|,
    parentDoc)
66 Else
67   parentDoc.RankedDocs = entryDoc.RankedDocs
68 End If
```

It is worth mentioning that although the duplicate UNIDs are removed (using @Unique), there is no attempt to cut the rate stack down to the size specified in the profile document. That will happen the next time the user rates a document.

---

## Views that show rated documents

Once you have the rating information, it makes sense to give new meaning to views by sorting documents based on the rating average. The more important, relevant, useful, and interesting documents bubble up to the top of the category where users can find them.

The FAQs view is an example of how to sort documents by average rating and then by the date the document was created:

FAQ ^		
<b>▼ Application Development</b>		
3	👍 01/19/2004 11:56:26 AM	LotusScript: Rich t
1	👍 01/19/2004 11:57:33 AM	Using the object-o
1	👍 01/19/2004 10:44:49 AM	LotusScript: Progr
<b>▼ LotusScript</b>		
3	👍 01/19/2004 11:56:26 AM	LotusScript: Rich t
1	👍 01/19/2004 10:44:49 AM	LotusScript: Progr
<b>▼ Troubleshooting</b>		
3	👍 01/19/2004 11:56:26 AM	LotusScript: Rich t

- **Column 1**

- Categorized, sorted in ascending order.
- Formula (change to match your documents):

```
Categories
```

- **Column 2**

- Hidden, sort in descending order.
- Formula:

```
@If(Average = ""; 0; Average)
```

- **Column 3**

- Displays the icon for the average rating.
- Formula:

```
@If(Average=0 | !@IsAvailable(Average); "";  
@Select(@Round(Average); 151; 152; 153; 154; 155))
```

- **Column 4**

- Secondary sort (descending) to rating.
- Formula:

```
PostedOn
```

If a document has not been rated, there will be no Average field, and it will fall below the documents that have been rated.

---

## Summary of design elements

To use the document rating strategy in your application, copy these design elements from

the example database to your database:

- Forms: ProfileDoc, UserActivity
- Subforms: Rate (embed in your content document forms as per the InfoDoc form), RankingbyImportanceDialog, RankingbyInterestDialog, RankingbyRelevanceDialog, and RankingbyUsefulnessDialog
- Views: ToRate, UserVisits, (copy columns 2 and 3 from the FAQs view)
- Agents: RankingDailyTally, RankingHourlyTally, DailyUserActivity
- Script Library: Ranking, UserActivity

Manually copy code from the Database script Options and Postopen events to those same events in your database. Finally, create your own version of the HomeFrameset frameset and HomeOutline outline control to give DBAdmins (only) the ability to open the profile document and hidden views.

**Note:** The InfoDoc form and FAQs view are not needed in your application, as they are used here just for demonstration purposes to represent any ordinary content document and view. By the way, the example application contains a few FAQs with content copied (and stripped down) from the [LDD Today](#) database.

---

## Conclusion

While full-text search does a great job of finding information buried in documents, there is no guarantee that what the search returns will actually be useful or truly relevant to the user. This article described an application strategy that actively involves users in the process of enriching document content by allowing them to rate each document's importance, relevance, usefulness, or interest. Document content is protected through the use of agents to update the ratings; users are prevented from rating the same document multiple times, thus ensuring the integrity of the rating process itself.

And because you are adding this feature to a Notes application using Domino Designer, it takes very little effort to quickly realize significant benefits.

---

## About the author

Kent Kurchak is the owner of [wareSource](#), an IBM Business Partner that provides robust, customizable, and economical training materials and applications for Lotus Notes and Domino users, developers, and administrators. wareSource courseware is used worldwide by IBM Business Partners, training companies, independent instructors/consultants, corporate training departments, and individuals.