LDD Today

developerWorks

Lotus. software

LotusScript: Programming views in Notes/Domino 6

Level: Advanced
Works with: Notes/Domino
Updated: 01-Aug-2003

by Sally Blanning DeJean
and David DeJean

This article is the fifth in a series that takes a close look at the new classes and enhancements to the LotusScript programming language in Notes/Domino 6. Previous articles dealt with the new LotusScript classes for manipulating rich text elements ("LotusScript: Rich text objects in Notes/Domino 6"), LotusScript support for XML ("LotusScript: XML classes in Notes/Domino 6" and "LotusScript: More XML classes in Notes/Domino 6"), and enhancements for programming administration functions ("LotusScript: The NotesAdministrationProcess Class in Notes/Domino 6").

In this fifth article, we look at the enhancements to the NotesView, NotesViewColumn, and NotesUIView classes. As with the previous articles, this one assumes that you are an experienced Domino application developer with knowledge of the LotusScript programming language.

New functionality for views

Many of the new view and view column methods and properties in the Notes/Domino 6 LotusScript classes NotesView, NotesViewColumn, and NotesUIView seem aimed at helping the harried developer who must constantly respond to user requests for changes to the views in their applications. These new and enhanced methods and properties allow developers to create scripts that non-developers can run to make changes to views without the intervention of the developer:

- There are seven new methods and five new properties in the NotesView class, and several other properties that were formerly read-only but are now read-write. Among the new features are the ability to lock the design of an element and to prohibit design refresh so the view design cannot be overwritten. It is now easier to count the number of documents in a view, and you can create scripts to allow users to change view selection formulas and to create, copy, or remove columns.
- The NotesViewColumn class has 17 new properties and 30 existing properties that have been expanded from read-only to read-write. You can use these to write scripts to change header styles (the font, point size, and color); to change column text and formatting, sorting, and hiding columns; and even to change the column formulas.
- The NotesUIView class has three new properties and two new methods, but most important are its three new events, especially the InViewEdit event.

These new features of LotusScript allow views to be altered by scripts in actions, agents, or view events. If you work in an organization where users frequently request changes to the selection criteria for views, the addition or removal of a column, a change to the appearance of a column, or a change to a column formula, you'll want to look closely at these new Notes/Domino 6 capabilities because they can probably make your life a lot easier.

Two other new capabilities greatly expand the way users will interact with views. The first allows non-designers to create customized shared views: A user can run a script to create the custom view she wants with the columns and formatting options she wants. The other uses the new InViewEdit event to allow users to create and edit documents directly in a view.

In this article, we'll explore five example applications that use these new LotusScript features:

- Programming the view selection formula
  Agents run from the action bar can rewrite the selection formula then redisplay the view. This can be a powerful tool for helping users customize a view without constant intervention by a developer.
- Programming the view's look and feel
  In the same way that you can program the view selection formula, you can program other attributes of the view's display—column formulas and display formats are two examples.
- Creating a new view
- Using InViewEdit
- Development aids
  Some of the new features of LotusScript can help developers work more efficiently. Locking design elements ensures that you and no one else can work on a design element, for example. You can enforce your company's design standards by running an agent that standardizes all the views in an application—setting the font, size, and color of header and column text—just before you release an application for testing.

You can download a Notes database named LSViews.nsf from the Sandbox that includes working code for these examples. The database can be run locally or from a server, but you need to open and resave all the agents in order to resign them with the appropriate ID and to set security to meet the requirements of your organization.

Applications that use these LotusScript features are in effect redesigning the database, so users either need Designer rights or permission to run the agents on behalf of a signer with sufficient rights. The easiest way to do this is to set up a group that includes all the users and to give the group on behalf of rights. For more information, see the *LDD Today* article, "Decoding the new Notes/Domino 6 Agent Features."

Example 1: Programming the view selection formula

Changing the view selection formula is a particularly useful example of giving users control that helps them to focus on information. In the LSViews database, a view named All presents an HR application: The documents list employees who are leaving a company, together with their department and last date of employment. The view displays several shared actions in its action bar. All of these actions execute agents stored in the application. (The reason for making them agents rather than actions is that actions cannot be run on behalf of.) One of these agents, Date Range, allows the database user to enter beginning and ending dates and to reset the selection formula for the view to display only documents that fall within that date range. The advantage of this agent over a generic full-text search is that the view's structure remains the same. The view display maintains category headings and the order of the documents in the view, both of which would be lost in a search.

The Date Range action uses an @Command formula to run an agent named Dates: @Command([RunToolsMacro];"(Dates)"). (The name of the agent appears in parentheses because it is hidden from the agent menu.) The agent prompts the user to enter starting and ending dates, then creates a selection formula from the values the user enters:

```
Sub Initialize
    'Prompts user for beginning and ending dates
    'Changes selection formula to display docs between dates
    Dim uiw As New NotesUIWorkspace
    Dim uiview As NotesUIView
    Dim view As NotesView
    Dim formula As String
    Dim one As String
    Dim two As String
    Dim ct As Integer
    one = uiw.Prompt(Prompt_OKCANCELEDIT,"Enter the starting date","Start Date (use format MM/DD/YY)","")
    two = uiw.Prompt(Prompt_OKCANCELEDIT,"Enter the ending date","End Date (use format MM/DD/YY)","")
    formula = |SELECT Date > [| & one & |] & Date < [| & two & |]|
    Set uiview = uiw.CurrentView
    Set view = uiview.View
    view.SelectionFormula = formula
    Call uiw.ViewRebuild
    ct = view.EntryCount
    Messagebox "The number of docs in this view = " & ct,,"Count of docs in view"
End Sub
```

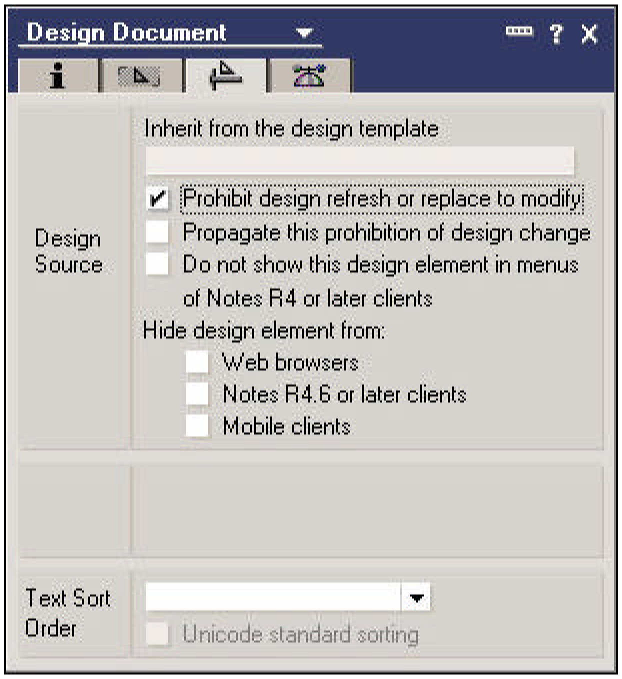
The new selection formula is created as a string named formula, which replaces the current selection formula in the line

```
view.SelectionFormula = formula
```

that uses the new NotesView property SelectionFormula to set the formula in the view. Then another new method of NotesUIWorkspace—ViewRebuild—rebuilds the view from the back-end (where it was created) and displays it immediately.

(Notice that the last line of the agent uses the new EntryCount property of the view to display a count of the documents in the view. Contrast this quick way of counting the documents in the view to the pre-Notes/Domino 6 way of looping through the view to count the documents. It is much easier and requires less coding.)

This script view action permanently sets the view selection formula until it is changed either by a designer or by running the view action again. It is important to keep this in mind if a design template is running on the server because it will overwrite the view when it refreshes. However, it is possible to prohibit a design refresh by using the new NotesView property IsProhibitDesignRefresh. IsProhibitDesignRefresh is a Boolean. The design cannot be changed if the "Prohibit design refresh or replace to modify" option is selected, as shown in the following.



Design Source
Inherit from the design template
[✓] Prohibit design refresh or replace to modify
[ ] Propagate this prohibition of design change
[ ] Do not show this design element in menus of Notes R4 or later clients
Hide design element from:
[ ] Web browsers
[ ] Notes R4.6 or later clients
[ ] Mobile clients

Text Sort Order
[ ] Unicode standard sorting

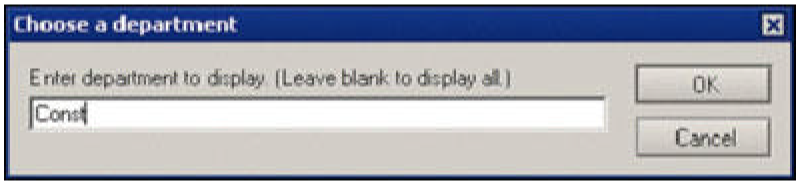
However, be forewarned—it can be changed with LotusScript. The By Dept. agent alerts the user if the property is checked and asks whether or not to remove the prohibition. If he chooses to do so, the line

```
view.IsProhibitDesignRefresh = False
```

allows the user to overwrite the view with his new choices. Of course, the user has to either be listed in the ACL as a Designer or be running the agent on behalf of someone who has rights to make design changes. The code for this agent is as follows:

```
Sub Click(Source As Button)
    'Changes selection formula to professions user chooses.
    Dim uiw As New NotesUIWorkspace
    Dim uiview As NotesUIView
    Dim view As NotesView
    Dim formula As String
    Set uiview = uiw.CurrentView
    Set view = uiview.View
    If view.IsProhibitDesignRefresh Then
        Messagebox "Design Refresh not allowed",,"The view does not allow changes."
        askchange = uiw.Prompt(PROMPT_YESNO,"Change design refresh settings?", _
        "Do you want to change the prohibition to change this view design?")
        If askchange = 1 Then
            view.IsProhibitDesignRefresh = False
        Else
            Exit Sub
        End If
    End If
    one = uiw.Prompt(PROMPT_OKCANCELEDIT,"Choose a department","Enter department to display. (Leave
    blank to display all.)","")
    formula = |SELECT @Begins(Profession;"|& one & |")|
    view.SelectionFormula = formula
    Call uiw.ViewRebuild
End Sub
```

This agent uses the contents of the field Profession to select documents for display. It prompts the user to enter the beginning of a department name or job title:


Choose a department

Enter department to display. (Leave blank to display all.)

Const

It uses this string in the selection formula as part of an @function—in the case of the example in the screen above, SELECT @Begins(Profession; "Const").

Two other actions rewrite the view selection formula in similar fashion. The Marketing Only agent limits the view to employees whose Profession field begins with Marketing by writing a selection formula:

```
    formula = |SELECT @Begins(Profession;"Marketing")|
```

The Show All agent resets the view to display all documents in the database with the selection formula:

```
    formula = |SELECT @all|
```

Example 2: Programming the view's look and feel

Just as you can program the view's selection formula, the new features of LotusScript allow you put other attributes of the view under program control as well. In the view named All, these features are used in four actions that change the formula and formatting of the first column in the view.

The two agents Small Display and Large Display change the typeface display of the employee names. Large

Display resets all four attributes of the typefont used in the first column of the view—typeface, size, style, and color—by writing to four attributes of the column specified by view.Columns(0):

```
Sub Initialize
	'Sets view to a standard color and font
	Dim uiw As New NotesUIWorkspace
	Dim uiview As NotesUIView
	Dim view As NotesView
	Dim vc As NotesViewColumn

	Set uiview = uiw.CurrentView
	Set view = uiview.View
	'Set first column to bold and 12 pt
	Set vc = view.Columns(0)
	vc.FontFace = "Default Sans Serif"
	vc.FontPointSize = 12
	vc.FontStyle = VC_FONT_BOLD
	vc.FontColor = COLOR_BLACK
	Call uiw.ViewRebuild
End Sub
```

For the complete lists of available values for FontStyle and FontColor, see the Domino Designer 6 Help. The FontFace value can be any name in the fonts list in the column Properties—just be careful to pick fonts that are installed on the computers that will run your application. The colors are not limited to the 16 named color constants. You can use the new NotesColorObject to set the font color—see the section titled "Setting More than 16 Colors" in "LotusScript: Rich text objects in Notes/Domino 6," a previous article in this series.

The Small Display agent is virtually identical to Large Display except for the values of the typeface attributes:

```
	vc.FontFace = "Default Sans Serif"
	vc.FontPointSize = 10
	vc.FontStyle = VC_FONT_PLAIN
	vc.FontColor = COLOR_BLACK
```

Notice that all four attributes of the text style are reset, even though some are not changed. This is because settings that aren't explicitly changed default to their current values, and if you don't specify all the settings, you may get some unexpected results.

Last/First and First/Last work in similar fashion to rewrite the column formula to change the display order of the names. The Last/First agent writes a formula for the first column in the view that concatenates the contents of fields that hold the employees last and first names, separated by a comma and a space:

```
Sub Initialize
	Dim uiw As New NotesUIWorkspace
	Dim uiview As NotesUIView
	Dim view As NotesView
	Dim vc As NotesViewColumn

	Set uiview = uiw.CurrentView
	Set view = uiview.View
	'Sets first column to display Lastname, Firstname
	Set vc = view.Columns(0)
	vc. Formula = |Last + ", " + First|
	Call uiw.ViewRebuild
End Sub
```

The First/Last agent reverses the formula order:

```
    vc. Formula = |First + " " + Last|
```

If you try the action buttons in the All view, you notice that the various look-and-feel actions don't overwrite each other. You can configure the view to show just the employee names in the Sales department in large text and lastname/firstname order. There are times when taking a more global approach to setting view attributes can be useful, as well: See the section "Enforcing Design Standards" later in this article.

Example 3: Creating a new view

You can use LotusScript in Notes/Domino 6 to add columns to an existing view or even to create a new view. In this example, an agent called "1. Create View and Add Columns" builds a view from scratch, allowing the user to pick the columns in the order he wants. This script requires the designer to create a master view that holds all the columns that will be available in the agent. In this case, the view is called Main, and it contains a column for each of the fields in the form Termination. The columns in the view do not have to be in any particular order, and the view can be hidden, so that it doesn't appear in the list of views available to the application's users.

In the Initialize subroutine shown below, the script presents an input box for entering a new name for the view the user is creating. Beginning at the line

```
Redim Preserve checkviewname(Ubound(db.Views))
```

it then checks the existing view names to verify that the new one is not already in use. If so, the script returns to the CHOOSENAME label and prompts again for a new view name. The line

```
Set UsersView = s.CurrentDatabase.CreateView(newviewname)
```

uses the new CreateView method of the NotesDatabase class. Because all views are created with one default column that contains the doc number, the line

```
Call UsersView.RemoveColumn(UsersView.ColumnCount)
```

removes that column with the new RemoveColumn method of the NotesView class. The script then finds all the columns in the Main view and creates an array, called allcols, with their header titles. The final Do Loop While loop creates a prompt that displays the array allcols as a list of columns for the user to choose from. The loop continues as long as the user indicates that he wants to add another column to the new view.

When the user makes a selection for a column in the new view, the script uses a For-Next loop to go through the column titles in Main to locate the correct column. When the script locates the column, it then copies that column to the new view. The line

```
Set col2 = UsersView.CopyColumn(col1, K)
```

uses the new method CopyColumn to do the copying. Col1 refers to the column in Main, and K is the column position in the new view. K increments as the user adds columns to the new view, so each new column is positioned after the previous column. Otherwise, all columns would be added to the first column position in the view, moving existing columns to the right.

The script then prompts the user for additional columns and goes through the For-Next loop again finding the column position in the Main view and copying the column to the new view. This continues until the user indicates at the prompt that she does not want to add more columns to the new view.

The If statement beginning

```
If col2.Position = 1 Then
```

checks for the column's position in the new view. If the column is the first one, the user is prompted to decide whether or not to sort that column. After the If statement, the line

```
col2.HeaderFontColor = COLOR_BLUE
```

sets all column headers to the color blue. See the "LotusScript: Programming views in Notes/Domino 6 sidebar" for the code example.

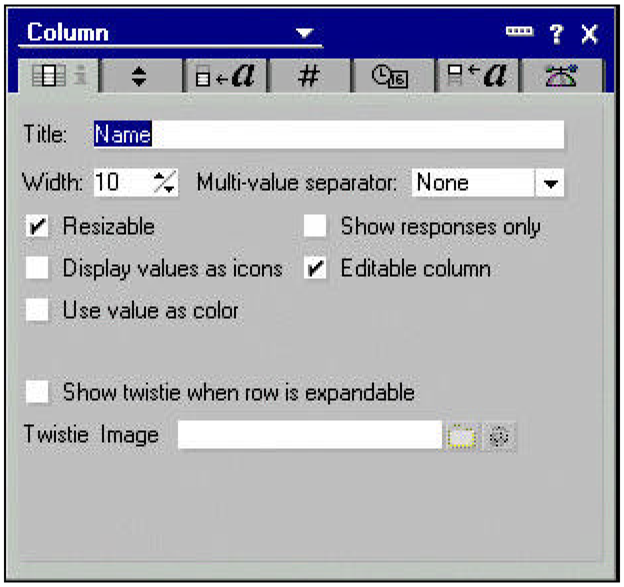
Example 4: Using InViewEdit

The new InViewEdit event allows users to edit or create documents directly in a view without opening the underlying document. This can be very handy. In-view editing would not be satisfactory for filling in fields that require several checkboxes or radio buttons, and it even has limits for lists, but it can be useful for rapid data entry or for quickly editing a field that needs a value changed.

For example, if a new field is added to a form, and someone has to go through all the existing documents updating that field, it may be faster and easier to do it in a specially created view rather than opening, editing, and saving each document. Or if someone is creating a series of new documents that contain only a few simple fields, using a view to create the documents could be preferable.

The InViewEdit event occurs when a user clicks once (double-clicking still opens the document) on an editable column or uses Ctrl + click to create a new document. It requires some setup in the view design.

The view must include at least one column with a field value assigned. On the Column Properties box Basics tab, select the Editable column option:



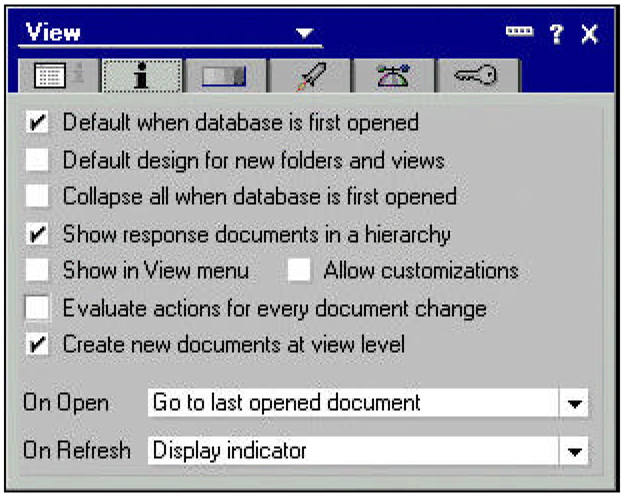
Title: Name

Width: 10 Multi-value separator: None

[✓] Resizable
[ ] Show responses only
[ ] Display values as icons
[✓] Editable column
[ ] Use value as color
[ ] Show twistie when row is expandable

Twistie Image

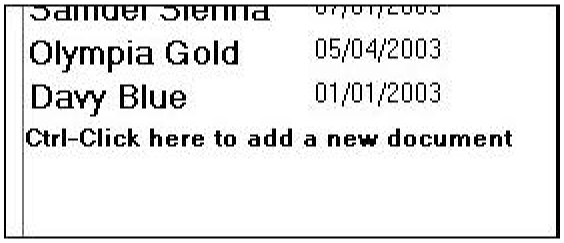
In the View Properties box Options tab, select the "Create new documents at view level" option. (If you only want users to be able to edit existing documents from the view, do not select this option.)



- [✓] Default when database is first opened
- [ ] Default design for new folders and views
- [ ] Collapse all when database is first opened
- [✓] Show response documents in a hierarchy
- [ ] Show in View menu [ ] Allow customizations
- [ ] Evaluate actions for every document change
- [✓] Create new documents at view level

On Open: Go to last opened document
On Refresh: Display indicator

When you open a view with this option selected, you see a line that says "Ctrl-Click here to add a new document" in the last row of your view:



Samuel Sienna	07/01/2003
Olympia Gold	05/04/2003
Davy Blue	01/01/2003

Ctrl-Click here to add a new document

If you press the Ctrl key while you click the mouse, an editing box appears in the editable column, allowing you to enter data. Even though the user can enter text in the column's editing box, nothing happens unless the designer has scripted the view's InViewEdit event. To use the InViewEdit event, you need to use and understand the parameters that follow. The basic syntax is:

InViewEdit(Source As NotesUIView, RequestType As Integer, ColProgName As Variant, ColumnValue As Variant, Continue As Variant)

By breaking it into its component parts, you can see how this event works:

- *Source As NotesUIView*
  Refers to the current, open view and contains all the properties and methods of the NotesUIView class.
- *RequestType As Integer*
  This integer indicates the kind of request being made in the view; request numbers are 1 through 4. One is reserved for query requests and is not available to use. A request type of 1 is automatically generated when a user clicks a column in a row to edit it or uses Ctrl + click to create a new document. Two is a request type to validate the field in the column, 3 is a request type to save a document, and 4 is a request type to create a new document.
- *ColProgName As Variant*
  Refers to the programmatic name of the editable column or columns. When the request is a type 1 or 2 (a query or a validation) the ColProgName is the current editable column in a view. As request type 1 or 2 ColProgName has only one element and therefore is always the array position numbered 0. When it is a request type 3 or 4, ColProgName is the programmatic name of the column currently being acted upon, starting with the array position number 0. IBM recommends that the programmatic name of the column (which can be found on the Advanced tab of the Column Properties box) be the same as the actual field name in the column. If it is not, you have to separately check each programmatic name before setting its value. (Notes automatically makes the programmatic name the same as the field name entered when you create the column, so in most cases you do not have to make changes.) Another issue to keep in mind is that if the programmatic name is not the same as the field name, the value may not even appear in your

opened document. For example, if you change a column field formula called Date to @Date(Date), the programmatic name changes to something like $2. You have to look for a programmatic column name of $2 in your script because it saves the value you enter in that column as a field called $2 and not in the field named Date.

- *ColumnValue As Variant*
  An array of the values in the editable column or columns referenced by ColProgName; the array defaults to string values and the array elements correspond one-for-one with ColProgName.
- *Continue As Variant*
  A Boolean that defaults to true, it can be set to false if a validation fails.

Creating an editable view

Let's take a look at a very simple InViewEdit event. You can try this in the LSViews database by opening the view named InViewEdit/Edit View 1. The script for this view is shown below. It performs only one request type—type 4—to create and save a new document and assumes there is only one editable column. In the first two lines, the script declares the NotesDocument and instantiates db, the database object. When the user executes the Ctrl + click command or when he clicks an existing editable column, the InViewEdit script is triggered. (Note: In this example, the editable column opens an editing box on an existing document, but it does not save the existing document. That requires a different request type.)

The script first creates a new document and gives the form field a name. Notes automatically finds the editable column, Colprogname(0), and gets its value. In this case, Colprogname(0) is equal to the programmatic name of the first column: Name. ColumnValue(0) is equal to the value the user types into the editable first column row.

The line

```
Call doc.ReplaceItemValue(Colprogname(0), ColumnValue(0))
```

then uses the ReplaceItemValue method of the NotesDocument class to replace the empty field that matches the column's programmatic name with the value the user typed in. The line

```
Call doc.Save(True, False, True)
```

saves the document.

```
Sub Inviewedit(Source As Notesuiview, Requesttype As Integer, Colprogname As Variant, Columnvalue As
Variant, Continue As Variant)

Dim doc As NotesDocument
    Set db = Source.View.Parent
    'Check for request to save a new document
    If Requesttype = 4 Then
        'Create a new document
        Set note = db.CreateDocument()
        'Give the doc form name
        note.Form = "TERMINATE"
        'Add the value to the document in the designated column
        Call doc.ReplaceItemValue(Colprogname(0), ColumnValue(0))
        'Save the document
        Call doc.Save(True, False, False)
    End If
End Sub
```

If the view has more than one editable column, replace the line

```
Call note.ReplaceItemValue(Colprogname(0), ColumnValue(0))
```

with a For loop, so the script loops through all the editable columns and fills in the values. See the example code in InViewEdit/Edit View 2:

```
For i = 0 To Ubound(Colprogname)
        'Add the value to document in the designated column
        Call note.ReplaceItemValue(Colprogname(i),ColumnValue(i))
Next
```

The only editing action a user can perform in this example view is to create and save a new document because request type 4 is the only request handled by the code. This may cause some confusion among users. Because they can edit any document in an editable view, they may expect changes to existing documents to be saved. If you want to deny users the ability to edit existing documents the best UI solution is to add an If statement that handles request type 3 by displaying an message box that spells out this restriction. You could do this by inserting the following code just after the If Requesttype = 4 statement in the example above:

```
Else
If Requesttype = 3 Then
        Messagebox "You cannot edit existing documents in this view", , "Error"
End If
```

Editing and validating fields
The Domino Designer Help that comes with Notes has several examples for InViewEdit scripts that use the Select Case statement to validate and edit as well as to create new documents and save them in the view. This coding technique breaks up the request types into blocks of easily read and managed code.

The script below illustrates the basics of handling all the available request types. You can find this example in InViewEdit/Edit View 3.

```
Sub Inviewedit(Source As Notesuiview, Requesttype As Integer, Colprogname As Variant, Columnvalue As
Variant, Continue As Variant)

        'Define constants for request types
        Const QUERY_REQUEST = 1
        Const VALIDATE_REQUEST = 2
        Const SAVE_REQUEST = 3
        Const NEWENTRY_REQUEST = 4

        Dim db As NotesDatabase
        Dim doc As NotesDocument
        Dim caret As String

        Set db = Source.View.Parent
        caret = Source.CaretNoteID
        If caret = "0" Then Exit Sub

        Set doc = db.GetDocumentById(caret)

        Select Case RequestType
        Case VALIDATE_REQUEST
        'Validate for empty column value
                If Fulltrim(ColumnValue(0)) = "" Then
                        Messagebox "Field has no value",, "You must enter a value in the column."
                        Continue = False
                End If

        Case SAVE_REQUEST
                For i = 0 To Ubound(Colprogname)
                        'Add the value to document in the designated column
                        Call doc.ReplaceItemValue(Colprogname(i),ColumnValue(i))
                Next
                'Save the document
                Call doc.Save(True, False, True)

        Case NEWENTRY_REQUEST
                Set doc = db.CreateDocument()
                'Give the doc form name
                doc.Form = "TERMINATE"
                For i = 0 To Ubound(Colprogname)
                        'Add the value to document in the designated column
                        Call doc.ReplaceItemValue(Colprogname(i),ColumnValue(i))
                Next
                Call doc.Save(True, False, True)
        End Select
End Sub
```

The four constants that begin the script are descriptive of the four request types handled in the InViewEdit event. It is not really necessary to include the QUERY_REQUEST line because currently it cannot be used by designers.

The lines referring to the caret are a reference to a document. CaretNoteID is a property of the NotesUIView. It is the document that is highlighted, or is being edited, in the current open view. The CaretNoteID is an alphanumeric string that identifies the document. If the caret returns a value other than zero, then the document exists and the editable columns can be validated and saved. If the caret returns a zero, then the document does not exist.

In that case, it means the user is creating a new document. Execution of the event script is stopped by the line

```
If caret = "0" Then Exit Sub
```

The view opens an edit box in the selected column and waits for the user to enter a value. The script is executed again when the user either tabs to the next column or presses Enter to indicate he is done with the document.

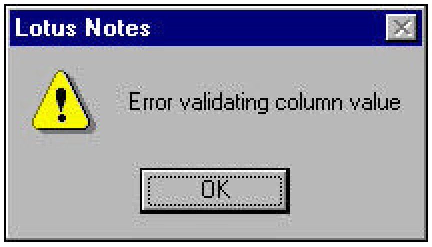
The VALIDATE_REQUEST case simply checks to make sure a value has been entered in the column's editing box. If no value has been entered, it displays a message requesting the user enter text. "Continue" is set to False to stop the event and return control to the user.

The SAVE_REQUEST is really a subset of NEWENTRY_REQUEST. It just doesn't create the new document.

Validating fields

Validating fields in in-view editing adds complexity to the code. The values edited in the view are always returned as text values. If, for example, you have a date field in the editable column, Notes treats whatever you enter in that column as text. However, you still want to make sure that the values entered are correct so that if you open the document later, or if you run a background agent to refresh fields, you do not get errors.

If you try to enter invalid text, Notes always throws up a message box like the one shown below on the validation error.


Lotus Notes

Error validating column value

In the following example, found in InViewEdit/Edit View 4, the Select Case statement for VALIDATE_REQUEST contains another Select Case statement—Select Case Colprogname(0). The Colprogname(0) corresponds to the column the user is editing at the time the validation occurs. In the following case, if the user has entered data in the first column, named Name, the script moves to the Case Name and checks for an empty column. If the user has entered data in the second column, named Date, then it validates using the Case Date.

In validation, the Colprogname index is 0. Zero is the column where the user is currently editing, and because only one column can be edited at a time for validation purposes, the currently edited column is always zero. Thus, the select statement is Select Case Colprogname(0).

```
Select Case RequestType
Case VALIDATE_REQUEST
    Select Case Colprogname(0)
    Case "Name"
    'Validate for empty column value
        If Fulltrim(ColumnValue(0)) = "" Then
        Messagebox "Field has no value",, "You must enter a value in the column."
            Continue = False
        End If
    Case "Date"
        If Isdate(ColumnValue(0)) Then
            Continue = True
                Else
                Messagebox "The value you entered is not a date",,"Date format 'MM/DD/YY' required"
            Continue = False
        End If
End Select
```

The first case, Name, checks for empty columns in the If statement beginning

```
If Fulltrim(ColumnValue(0)) = "" Then
```

and if it finds no value, it presents the message box and returns control to the user. When the user enters a value and presses the Tab or Enter key, the script resumes.

The second case, Date, checks to see if the entry is an acceptable date format and prompts the user to try again if it is not. (Even though the messagebox in the example specifies a MM/DD/YY format, MM/DD, M/D, and M.D are also accepted by the Isdate function.)

It is important to remember that even though the data may be entered in a date format it will be saved as a text string. The same is true of numbers: the Domino Designer Help includes an example of using IsNumeric to test for a number, but numbers also are saved as text strings if entered through a view. In many applications, text may be all that's needed, so you're finished. But in others the entry may need to be converted to another data type, which adds another layer of complexity to the script.

Converting dates and case
What happens if you have a list or a radio field that allows only one of a set of preselected values? In the following example, we take the field named Processed, a radio field, and force the user to enter one of the acceptable values. Unfortunately, you cannot display a picklist in InViewEdit, but you can validate the value, and, as in this case, proper case it as well.

You can see the code for this example in the InViewEdit event for the view InViewEdit/Edit View 5. In this example, the user can edit existing documents only. The second and third columns are both editable.

In the example InViewEdit/Edit View 4, the date could be tested for a legitimate date format, but it was still written into the document as a text value. In this example, the script converts the date entered in the second column into a true time/date field. In the third column, the user enters one of five values or is prompted if he enters an incorrect value. In addition, the value is put into proper case.

To prepare the view, add the line

```
%INCLUDE "c:\Lotus\Notes\lsconst.lss"
```

to the View's Global options. The LotusScript constants file contains the Public constant SC_ProperCase, which the script uses.

In the view's declarations, add Dim Pval As String and Dim Dval As Variant. Pval is a string value for the Processed field and Dval is a variant for the time/date field.

The validation for the Date column is almost the same as it was in the View 4 example. Note that the main difference is the addition of the line Dval = Cdat(columnvalue(0)). This line converts the value the user enters into a date and assigns it to the Dval variable of type variant.

```
    Case "Date"
        If Isdate(ColumnValue(0)) Then
            Dval = Cdat(columnvalue(0))
            Continue = True
            Else
            Messagebox "The value you entered is not a date",,"Date format 'MM/DD/YY' required"
            Continue = False
        End If
```

The Process field is handled differently. First a string array (named elements) of the five values the field can hold are created. Then in the Forall loop the script checks the value the user entered against the values in this elements array. The elements and the column value are converted to lowercase to make the comparison. If the column value does not match one of the elements, a message box displays a list of the values the user may enter, and he is returned to the edit box in the view.

If the column value is legitimate, it still must be converted to proper case in case the user did not enter it correctly. The last line converts the value to a text string and then proper cases that string and assigns it to the variable Pval.

```
    Case "Processed"
        'list of values user can enter in Processed editable column
        Dim elements(4) As String
        elements(0) = "Completed"
        elements(1) = "In Progress"
        elements(2) = "On Hold"
        elements(3) = "Not Started"
        elements(4) = "Discarded"

        'Check for a value not in "Processed" list
        Dim flag As Boolean
        flag = False
        Forall element In elements
            If Lcase(element) = Lcase(Columnvalue(0)) Then
                flag = True
        Exit Forall
            End If
        End Forall
            If Not flag Then
            'Create a list of values the user can enter in editable column and display to user
                Dim msg As String
                Forall element In elements
                msg = msg & element & Chr(10)
            End Forall
            Messagebox msg,, "Value must be one of the following"
            continue = False
        End If
        'Get value user entered as a string
        cval$ = ColumnValue(0)
        'Convert the string to propercase
        Pval = Strconv(cval$,SC_ProperCase)
```

If the Pval and Dval variables do the conversions in request type 2, they must be created in the Declarations because they must persist until the values are saved to the document. They do not need to be global if converted during request type 3 or 4. When it comes time to save the document, the current column value (i) is compared to the Pval variable. If it is a match, then the column value is changed to the Pval value and added to the document.

The date column doesn't use the same kind of comparison because there is no way to do a true match—Pval is a variant and the column value of the date the user entered is a string. Here the script checks the Colprogname, and if it is Date, then it gets the Dval variable and replaces the column value with it.

```
Case SAVE_REQUEST
    For i = 0 To Ubound(Colprogname)
        'Add the value to document in the designated column
        'If the column value matches value in Pval variable, it is used instead of the actual column value
        If Lcase(ColumnValue(i)) = Lcase(Pval) Then
            ColumnValue(i) = Pval
            Call doc.ReplaceItemValue(Colprogname(i),ColumnValue(i))
        Else
            If Colprogname(i) = "Date" Then
            ColumnValue(i) = Dval
            Call doc.ReplaceItemValue(Colprogname(i),ColumnValue(i))

            Call doc.ReplaceItemValue(Colprogname(i),ColumnValue(i))
            End If
        End If
        Next
        Call doc.Save(True, False, True)
```

These examples show the basic building blocks for in-view editing. As you develop more complex code, keep in mind that the script starts evaluating every view-editable field in the row starting with the column clicked and going to the right as the user presses the Tab key. If your view has 12 columns and all are editable, the InViewEdit event processes all 12 columns. If you want to allow a user to edit only one field from a view, do not make other fields in the row editable too.

Example 5: Development aids

Some of the new features of LotusScript can help you work more efficiently as you design your application. Here are two examples—locking design elements, which ensures that you and no one else can work on a design element, and using the new access to column properties as a way to automate compliance with design standards.

Locking a design element

Notes/Domino 6 allows designers to lock three kinds of design elements—agents, forms, and views. Locking prevents those who are not listed as lockholders from changing the element. If Peter Purple has a lock on the view design and he is the only lockholder, then he is the only one who can make design changes and save them as long as the lock is enabled with his name. More than one person or group can be listed as lockholders. If Peter Purple and Brian Blue are both listed as lockholders, then either one of them can unlock the design. A lockholder must disable the lock to let someone else edit and save the design. Locking writes the name of the current lockholder to the $Writers field and the date the lock was made to the $WritersDate field of the design element.

In order for locking to work, the database must be assigned an Administration server in the ACL. This server is also referred to as the Master Lock Server. If the database designers are all working from a network server, and not offline, locking can occur. If a designer is working offline, a provisional lock is placed on the design element. At the time the database is replicated back to the server, if no one else has tried to make a change, the changes the provisional lockholder made go into effect.

(You can also lock and unlock a view, agent, or form in Domino Designer using Lock Design Element and Unlock Design Element on the Design menu. You see a padlock icon beside locked design elements. If you are already in Domino Designer, using this menu is probably easier than running an agent. However, using the Design menu only allows you to lock or unlock an element that holds your own name as lockholder—unless you are the manager of the database. You cannot lock the design element with someone else's name or with a group or a list of names unless you run a script.)

An example of an agent that performs locking and unlocking is in the agent script below, titled Lock-Unlock Design Element. This agent checks to make sure that the design element can be locked (that is, that design refresh or replace has not been prohibited on the design element). Then it checks to determine if the element is locked. If so, it displays a message box with the name or names of the lockholder. If the lockholder is also the current user and the element is locked, it prompts the user asking if he wants to unlock the element. If the element is not locked, the script prompts the user asking if he wants to lock it.

A new Notes/Domino 6 property, IsDesignLockingEnabled, is used in the line

```
db.IsDesignLockingEnabled = True
```

This property is a Boolean. Setting it to false keeps a design element from being locked. The NotesView class has a new property called LockHolders, an array containing the names of the groups or people who have the right to lock or unlock the design element. The line

```
If Lholders(0) = "" Then
```

checks the LockHolders field. If it has no value, then the design element is not locked, and the script prompts the user, who must be at least a Designer in the ACL, to lock the database by entering the names of the persons who will have the rights to edit or unlock the design element. If the user enters one or more names when prompted in the line

```
qu2 = uiw.Prompt(PROMPT_OKCANCELEDIT, "Lock Holders?", "Who should the lock holders be? If you leave the box blank your name will be added as the lockholder.")
```

the line Call view.Lock(qu2,True) locks the element with the names of the Lock Holders, and no one else can edit it.

If the element had been locked by the current user and he wishes to unlock it, the line Call view.UnLock unlocks it. If the design element is locked and the current user did not lock it, he sees only the message box with the name of the current lockholder.

```
Sub Initialize
	Dim session As New NotesSession
	Dim uiw As New NotesUIWorkspace
	Dim uiview As NotesUIView
	Dim db As NotesDatabase
	Set db = session.CurrentDatabase
	Dim view As NotesView
	Dim cun as String
	Dim qu as String
	Dim qu2 as String
	Dim LH as String
	Dim ans as String

	db.IsDesignLockingEnabled = True
	cun = session.CommonUserName
	On Error Goto errh
	'Get view
	Set uiview = uiw.CurrentView
	Set view = uiview.View
	'Check if view design is locked
	Lholders = view.LockHolders
	If Lholders(0) = "" Then
		Messagebox "There are no locks on the design of this view",,"Status of locking"
		qu = uiw.Prompt(PROMPT_YESNO,"Change status?","Do you want to lock this view design?")
		If qu = 1 Then
			qu2 = uiw.Prompt(PROMPT_OKCANCELEDIT, "Lock Holders?", _
			"Who should the lock holders be? If you leave the box blank your name will be added as the
			lockholder.")

			Call view.Lock(qu2,True)
			Messagebox "The view name " &view.Name &" is locked",,"Lock status"
			Else
			Exit Sub
		End If
	End If

	'Recheck view holders
	Lholders = view.LockHolders
	Forall L In Lholders
		LH = LH & L & Chr(13)
	End Forall
	Messagebox "The lockholders on the design of this view are " & Chr(13) & LH,,"Status of locking"
	Forall L In Lholders
		If Trim(L) = cun Then
			ans = uiw.Prompt(PROMPT_YESNO,"Change status?","Do you want to unlock this view design?")
			If ans = 1 Then
				Call view.UnLock
			Else
			Exit Sub
		End If
	End If
	End Forall
errh:
	If Err() = lsERR_NOTES_LOCKED Then
		Print "View NOT locked - " & view.Name
		Else
			Messagebox "Error " & Err() & ": " & Error(),, "Error"
	End If
	Exit Sub
End Sub
```

Enforcing design standards

If you work with other designers, or even if you work alone, you may find it tedious to make your views look consistent within an application or to adhere to company standards. Using new LotusScript methods and properties, you can write an agent that standardizes the design of all the views in an application—setting the background color, font, size, and color of header and column text, even column width—just before you release an application for testing. This was not possible in previous releases of Notes/Domino because these characteristics were read-only. Now they are read-write.

To see how this works, try the two agents 4. Set View Type 1 and 5. Set View Type 2. These agents work on just the current view. The code is very similar to the script for Large Display and Small Display (see the earlier section "Example 2: Programming the view's look and feel"), except that it adds additional lines that set the background color for the view and modify the typefont used in the column headers.

What's missing is a way to run the agent on all the views in the database, and that is provided in the agent 6. Standardize All Views below. It sets a variable, I, that is the number of views in the current database, then runs a For loop to select each view in turn and to rewrite the values of the view's background color and column display attributes:

```
Sub Initialize
    'Loops through all views and sets column and header attributes to standards
    Dim uiw As New NotesUIWorkspace
    Dim view As NotesView
    Dim vc As NotesViewColumn
    Dim db As NotesUIDatabase

    Set db = uiw.CurrentDatabase
    'Get number of views in database
    Dim J As Integer
    Dim i As Integer

    For J = 0 To Ubound(db.Database.Views)
        'Begins changing each view
        Set view = db.Database.Views(J)

    'Set view background color
        view.BackgroundColor = COLOR_WHITE

    'Set first column to bold and 12 pt.
        Set vc = view.Columns(0)
        vc.FontFace = "Default Sans Serif"
        vc.FontPointSize = 12
        vc.FontStyle = VC_FONT_BOLD
        vc.FontColor = COLOR_BLACK

    'Set all other columns to plain and 10 pt.
        For i = 1 To Ubound(view.Columns)
            Set vc = view.Columns(i)
            vc.FontFace = "Default Sans Serif"
            vc.FontPointSize = 10
            vc.FontStyle = VC_FONT_PLAIN
            vc.FontColor = COLOR_BLACK
        Next
    'Set all headers to same font color, face, and point size.
        For i = 0 To Ubound(view.Columns)
            Set vc = view.Columns(i)
            vc.HeaderFontFace = "Default Sans Serif"
            vc.HeaderFontPointSize = 10
            vc.HeaderFontStyle = VC_FONT_BOLD
            vc.HeaderFontColor = COLOR_DARK_BLUE
        Next

    Next
    Call uiw.ViewRebuild
End Sub
```

Notice that again all four font attributes are set for both the column text and header text, just to prevent surprises. When all the views have been processed, the current view is rebuilt to display the changes. If the database includes views that you don't want to standardize, you can set the Prohibit design refresh option for those views or write your agent to prompt you for permission before it resets a view's attributes.

Conclusion

These examples are only an introduction to the changes in LotusScript for Notes/Domino 6. There are many more attributes of views and columns that you can program because of the change in properties from read-only to read-write in the NotesView and NotesViewColumn classes. The InViewEdit event in the NotesUIView class adds new rapid editing capabilities. These changes mean that in Notes/Domino 6 you gain ability to manipulate not only the data in views but also the UI of views. For the first time, you can allow non-designers to make changes in views—a good thing for Domino developers because it holds the promise of considerably reducing the drudgery of application maintenance.

ABOUT THE AUTHORS

Sally Blanning DeJean and David DeJean have been working with and writing about Lotus Notes and Domino for as long as they've existed. They were co-authors of the very first book about Notes, *Lotus Notes at Work* . Sally, a CLP Principal, has written other books about Notes and is a full-time Notes/Domino developer. David, a CLP, has been an editor and writer for several computer publications. He is a partner in DeJean & Clemens, a firm that develops Notes and Internet applications and technical and marketing communications.

LotusScript:Programming views in Notes/Domino 6 sidebar

The following is a code example for creating a new view.

```
Declarations
Dim db As NotesDatabase
Dim Templateview As NotesView
Dim UsersView As NotesView
Dim col1 As NotesViewColumn
Dim col2 As NotesViewColumn
Dim J As Integer
Dim K As Integer
Dim L As Integer
Dim allcols () As String
Dim checkviewname () As String

Sub Initialize
    Dim s As New NotesSession
    Dim ws As New NotesUIWorkspace

    Set db = s.CurrentDatabase
    Set Templateview = db.GetView("MAIN")
    'Asks for name of new view
CHOOSENAME:
    newviewname = Inputbox("Name for new view?","What do you wish to name this view?")
    If Trim(newviewname) = "" Then
        Messagebox "Stopping processing",,"You did not enter a name for the view."
        Exit Sub
    End If
'Check for existing view name
Redim Preserve checkviewname(Ubound(db.Views))
For L = 0 To Ubound(db.Views)
    checkviewname(L) = db.Views(L).Name
    If checkviewname(L) = newviewname Then
        Messagebox "The view name you entered is already being used. Please pick another." _
            ,,"View Name already in use"
            Goto CHOOSENAME
    End If
Next

    'Create new view with name from user input
    Set UsersView = db.CreateView(newviewname)
    'Remove default column created with view
    Call UsersView.RemoveColumn(UsersView.ColumnCount)
    'Find number of columns in Templateview
    I = Ubound(Templateview.Columns)

    Redim Preserve allcols(I)
    'Create an array of column titles to be used in prompt
    For J = 0 To I
```

```
        allcols(J) = Templateview.Columns(J).Title
        Next
        K = 1
Do
    chosencol = ws.Prompt(Prompt_OKCancelList,"Choose column to insert", _
    "Select from list of column names.","",allcols)
    'Find column position
    For J =0 To Ubound(Templateview.Columns)
    If  Templateview.Columns(J).Title = chosencol Then
    Set col1 = Templateview.Columns(J)
    End If
    Next
    Set col2 = UsersView.CopyColumn(col1, K)
    If col2.Position = 1 Then
        asksort = ws.Prompt(PROMPT_YESNO,"Sorting for first column?", _
        "Do you want to sort the first column?")
        If asksort = 1 Then
            col2.IsSorted = True
        End If
    End If
    'Set the font color for the header
    col2.HeaderFontColor = COLOR_BLUE
    K= K+1
Loop While ws.Prompt(PROMPT_YESNO,"Additional columns?","Do you want to add another column to the view?")
    Messagebox "View completed",,"Your view has been built."
End Sub
```