

by
[David](#)
[DeJean](#)

Level: Intermediate
Works with: Designer 5.0
Updated: 09/01/2000

"Hide-whens" have a real personality. They are one of the most important tools in a developer's kit for building a quality user interface for a Domino application. Hide-whens can improve the user interface (UI) of an application, smooth out display differences between Notes clients and Web browsers, and help custom-tailor document contents to their readers. But sometimes hide-whens seem to have a mind of their own. Judging from the number of postings to Notes.net discussions, hide-whens can sometimes be a cause of frustration.

This article explores the ins and outs of using hide-whens, discusses some examples, and offers suggestions for some of the problems you may encounter. It assumes a solid understanding of using Domino Designer R5 to develop Notes/Domino applications.

Hide-when basics

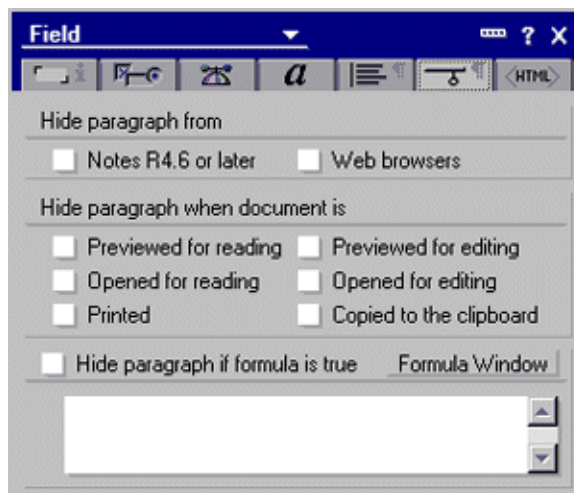
Frustration with hide-whens may arise, in part, because they require some reverse thinking. Most Notes formulas are by their nature "show-whens"—the formula displays something to the user. Hide-whens typically *prevent* the display of something. Another problem is that while developers tend to think of their application's UI object-by-object, hide-whens don't work object-by-object, they work paragraph-by-paragraph. And finally, hide-whens don't work the same way for all objects. Hide-whens typically apply to a field and its contents. But in a rich text field, for example, hide-whens can be applied to subsets of the field's contents. Changes in settings can ride into the field on pasted text or graphics. While this is usually the right thing for the content, it can be a surprise for you.

Hide-whens are invoked by the evaluation of a condition that takes the form **if (A) then hide (B)**. For example:

- **If** the document is in Read mode, **then hide** this field.
- **If** the field TotalSales is empty, **then hide** this field.
- **If** the client type is "Web," **then hide** this form.
- **If** the reader is not the next approver, **then hide** this button.

Hide-whens allow you to manipulate what appears in the UI of the application based on the state or properties or contents of the object the hide-when formula is applied to. Almost anything in the UI can be hidden—fields, buttons, text, or graphics on forms. (The major exceptions are rich text fields and tables, which are discussed later in this article.)

Hide-whens are invoked in the object's properties box; the Hide tab is the one with the window shade. The most common reasons for using a hide-when can be selected on this tab:



Check any of the boxes, and the paragraph containing the object will be hidden when that condition exists. The options on this tab have become more elaborate as Notes features have proliferated. It now allows you to:

- **Hide the paragraph from Notes or the Web** – These choices help enhance the application's compatibility with particular clients. You might hide a file-upload control from Notes clients, and hide a rich text field intended to hold file attachments from Web users.
- **Hide the paragraph when the document is previewed or opened for reading** – These choices can conceal elements when the document is in Read mode that are used only when the document is in Edit mode or used only by the application's developer.
- **Hide the paragraph when the document is previewed or opened for editing** – These choices can hide elements used only in Read mode. When used with the Read mode choices, they let you give users interface devices that help them enter data while the document is being edited, and then hide those interface devices so that the data is presented as simply as possible in Read mode.
- **Hide the paragraph when the document is printed or copied to the clipboard** – These two choices are gentle enforcers of security for sensitive information. (They work only in Notes clients, of course, not in Web browsers.) They don't make it impossible to extract data from a Domino application, but they make it more difficult.
- **Hide the paragraph when the formula is true** – This allows you to specify hide-when conditions that are specific to your application, using the full range of fields and properties. You can use @functions and formulas that base conditions on the ACL or the contents of other fields.

Three major uses for hide-whens

In Notes applications, there are three major uses for hide-whens. All of them are related to giving the application the best possible user interface, which means presenting appropriate information as simply and clearly as possible and helping the user understand the range of possible actions:

- **State-related** – Used when the designer wants to present a field in one format if the document is open in Edit mode, but in another format if the document is in Read mode.
- **User-related** – Used when content must be hidden from some users but displayed for others, based on user roles.
- **Client-related** – Used when a feature works on one client but not on another, for instance, rich text fields in the Notes client versus in a Web browser.

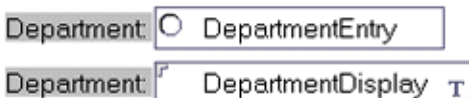
There are some alternatives to hide-whens for particular uses; computed subforms is one example. And finally, there are limits to what hide-whens will

hide, and how effectively they hide it. More on that later.

State-related hide-whens

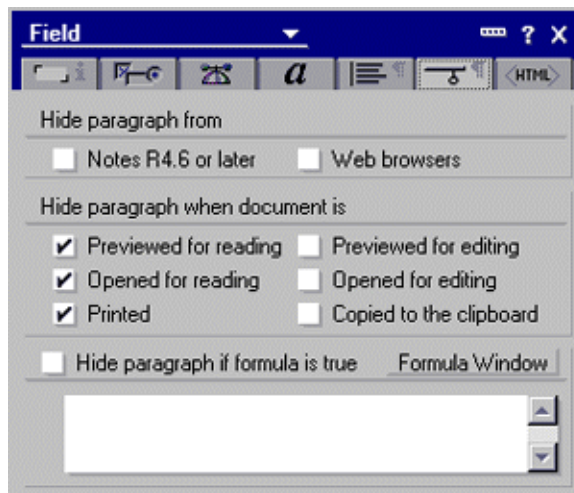
Perhaps the most common use for a hide-when is to give the user the gentle guidance of a UI control when they're entering data into a form, and then to hide that control when the saved document is opened for reading. You might want to give users a set of radio buttons to choose a department name when they're creating or editing a document, but just display the name of the selected department when the document is opened for reading.

To do this, you create two fields on the form, one directly above the other. In Designer, these fields look like this:

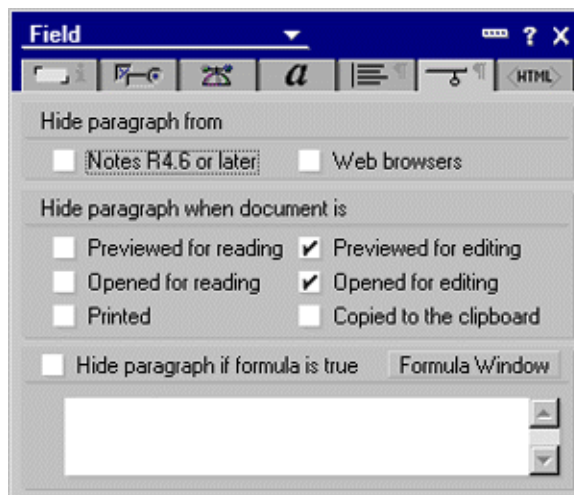


You create the DepartmentEntry field as an editable set of radio buttons. You set the DepartmentDisplay field to be plain text of type Computed for Display with its default value set to the field name DepartmentEntry.

On the Hide tab of the properties box for the DepartmentEntry field, select the hide-when options to keep the radio buttons from being displayed when the document is in Read mode:



Then select the DepartmentDisplay field and do the opposite; select the hide-when options Previewed for editing and Opened for editing:



Note: As an alternate to setting the field properties, you can achieve exactly the same affect by selecting each text label that says Department and setting its hide-when properties. Or you can select the entire paragraph and set its text properties. This can be a source of some confusion; just remember that when you set properties for a field or a label—any part of a paragraph—the settings affect the entire paragraph. In this case, the labels are on the same lines as the fields, so they're part of the paragraphs and take on the field properties.

The result of these selections gives users guidance in choosing one and only one department by using a panel of radio buttons when the document is being created or edited (as shown on the left below), but displays their choice much more simply and takes up much less space when the document is being read (as shown on the right):



This simple technique can make a big difference in the appearance of documents as users toggle between Edit and Read modes. A similar use of hide-whens can help reduce the clutter of UI gadgetry in documents being edited. If the data or selections can be categorized, you can let the user choose the category, and then display only the relevant items.

Here is an example of cascading selections that uses a dialog list field named InterviewSelector to let the user pick a subtopic within a Help Desk trouble report form. The fields for each subtopic are hidden until the user selects that subtopic—either from the InterviewSelector list or via a button that calls the next list choice.

First, create the dialog list field InterviewSelector and enter the names of the subtopics as the list choices. Be sure to select the "Refresh fields on keyword change" checkbox.

The screenshot shows the 'Field' dialog box with the following settings:

- Display:** ☒ Show field delimiters
- Choices:** Enter choices (one per line)
 - 1. User
 - 2. Problem type
 - 3. Description
 - 4. Hardware
 Buttons: Sort, List Window...
- Options:**
 - ☐ Allow values not in list
 - ☐ Look up names as each character is entered
 - ☒ Display entry helper button
 - ☒ Refresh fields on keyword change
 - ☐ Refresh choices on document refresh

To make the form open in the proper state (with the first item in the list selected), give the field this default value:

```
@If(@IsNewDoc; "1. User"; InterviewSelector)
```

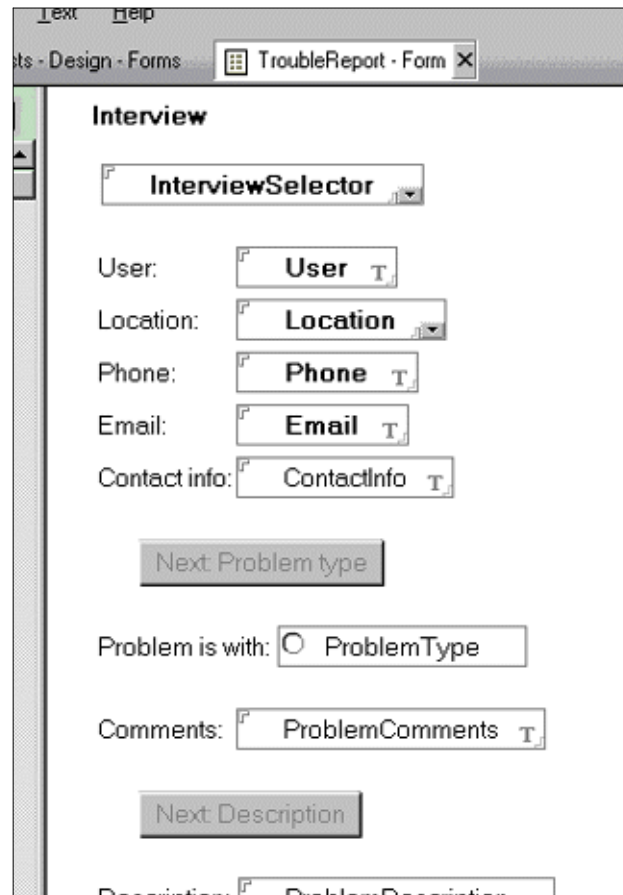
Then add the appropriate fields for each subtopic to the form, one below the other, spaced with blank lines as needed for appearance. Set the Hide formula for each of these fields (and any blank lines below it—this maintains alignment) so that they are hidden unless the corresponding subtopic is selected in the InterviewSelector field. For example, the Hide formula for the user information fields—for example, User, Location, Phone, Email, and ContactInfo—would be:

```
InterviewSelector != "1. User"
```

And the Hide formula for the problem type fields—for example, ProblemType and ProblemComments—would be:

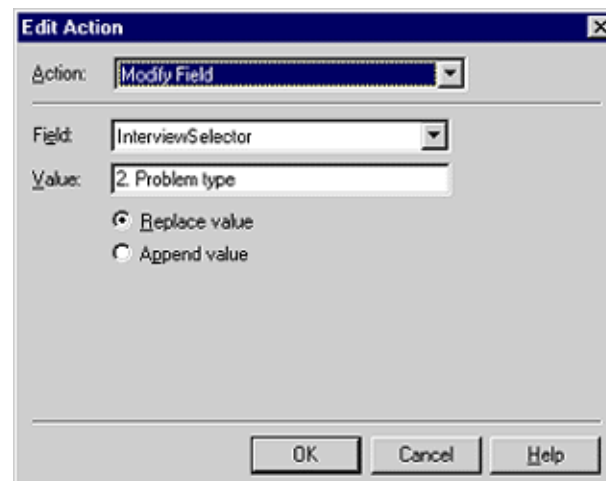
```
InterviewSelector != "2. Problem type"
```

A section of the form looks like this:

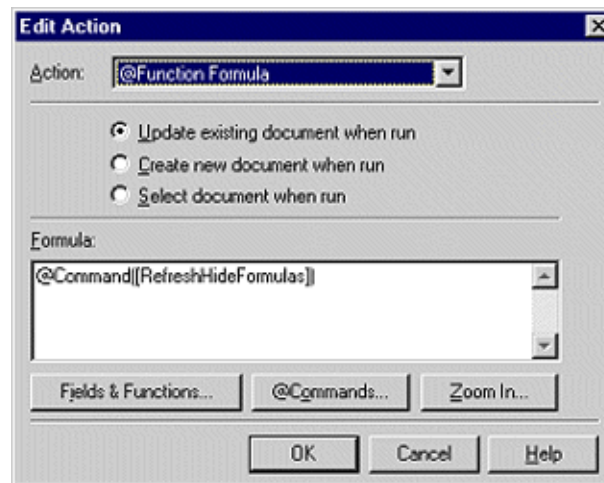


The buttons, like the one that says Next: Problem type, perform the same function as the InterviewSelector list—they reset the value of InterviewSelector to the next subtopic, and then refresh the Hide fields to reveal the next set of data fields. They provide the user with easy, orderly navigation of the form. The user can step through the form using the buttons, or pull down the InterviewSelector list to return to a previously visited section.

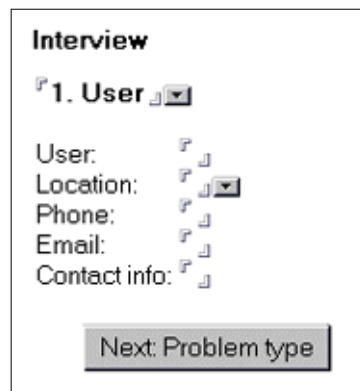
The button code consists of two simple actions created in the Click object. For the button labeled Next: Problem type in the illustration above, the first action resets the value of InterviewSelector to 2. Problem type:



The second action runs an @command that refreshes the document's fields so that the hide-when formulas are evaluated:



When the user creates a new document, the form opens on the first subtopic:



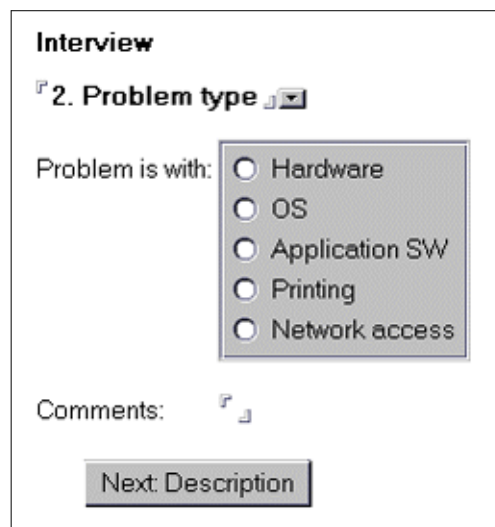
Interview

1. User

User: [icon]
Location: [icon]
Phone: [icon]
Email: [icon]
Contact info: [icon]

Next: Problem type

When the user completes the user information section and clicks the Next: Problem type button, the fields that are displayed for the user change to the second subtopic's fields:



Interview

2. Problem type

Problem is with:

- ☐ Hardware
- ☐ OS
- ☐ Application SW
- ☐ Printing
- ☐ Network access

Comments: [icon]

Next: Description

The hide-whens yield a visually simple, organized application that collects data into a single document. It guides users through the required steps in logical order and doesn't overwhelm them with a long, cluttered form they must scroll down to complete.

User-related hide-whens

Hide-whens are frequently used to give different groups of users different views of the data in a document, or to tailor a document to its current user. Here's the top of a form from a purchase-order system that uses workflow for approvals:

Proposer: Tim Pennington First level approver:
 Dept: Applications Engineering Second level approver:
 Use this PO for ongoing POs such as communications costs for Atlantic Bell, AT&T, etc.

When a user creates a new document, her name is automatically filled in. She selects the department the PO will be charged to from a drop-down list, and chooses first- and second-level approvers also from drop-down lists—a department director for the first level approval and a vice-president for the second.

Here is the design for the same section of the form:

POBody
 Ready
 Date submitted
 Submit
 PO# POno
 Proposer Director DirAmt
 Dept VP VPAmt
 Thirdapprover
 Link to Req Cancel PO

The simplicity of the visible UI contrasts with the complexity of the application design. Hide-whens are used extensively to maintain this simplicity, yet allow the application to be as flexible as possible. A document created with this form can have three states: it can be saved as a work-in-progress, it can be saved as ready to submit, or it can be saved and submitted. The save step is obviously crucial, and much of the programming in the form is keyed to the save action. Three of the fields on the right side of the table are filled in by queriesave operation: the POno field is a computed look-up into another database that returns the next available PO number, and the DirAmt and VPAmt fields are also populated by computed-when-saved lookups that enter the amount of money each approval has signing authority for. If the amount of the PO entered further down exceeds these limits then the Thirdapprover field—for a senior vice-president—is made part of the workflow and the user is prompted to select a third approver.

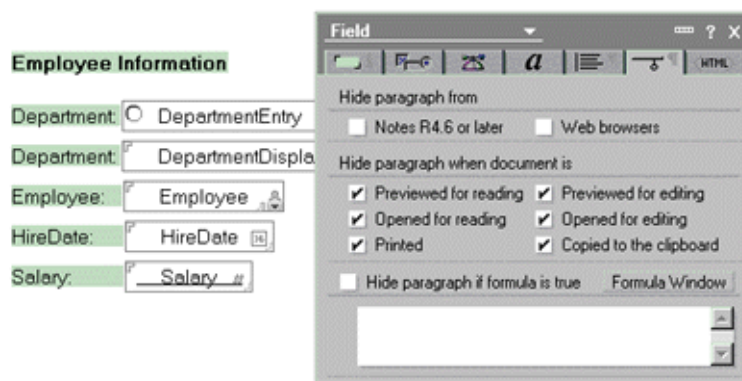
The top two rows and the bottom row of the table are hidden until the document is saved. If items and prices have been entered (lower down on the form) and the save action verifies that the PO is ready to submit, then the Submit button, the PO number, the Link to Requisition and Cancel PO buttons appear. (Link to Requisition, if clicked, lets the user choose a requisition document from a list of existing requisitions. A doclink to the requisition appears in the POBody field in the top row of the table.)

If the Submit button is clicked, the SubmitDate field is filled in, e-mail is sent to the first approver, and the Ready field, a flag field, is set to Yes. The hide-when formula on the Submit button won't allow the button to appear if Ready is Yes, which prevents a PO from being submitted more than once.

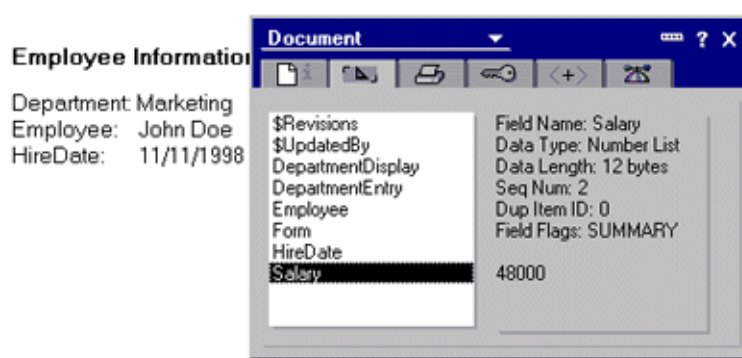
The table serves a valuable function here, because objects in individual table cells can be hidden or revealed. Otherwise the entire line—or paragraph—would be affected.

It's important to understand that using hide-whens to hide data removes it from the display, but it does not remove it from the document. All the fields in the document and their contents continue to be visible in the document's properties box.

For example, it's common for a developer to hide fields from both authors and readers by checking off all the properties box's hide-when options. You might do this for fields that hold temporary or intermediate values. But be careful. You may use sensitive information like salaries only for calculations, and hide it in both Edit and Read modes:



But the field still exists in the document, and even though it doesn't display in the document it is still reported in the Document properties box:



The moral of this story is this: don't depend on hide-whens alone. The only way to absolutely hide content in a Notes database is to set the Readers and Authors properties in the Security tab of the form's properties box, or add Readers and Authors fields to the document itself. The purchase-order application, for example, has a complex reader's field that restricts access to documents by department, based on department numbers, ACL groups, and individuals named in the document:

```
@If(DeptCode =
"ADFG":"ADFH";Proposer:VP:Director:CFO:"[President]":"[CEO]":"
Company Counselors":"Purchase Order Readers":"Tim Pennington";
@If(DeptCode =
"EMEM";Proposer:CFO:"[PresAdmin]":"[President]":"[CEO]":"Purchase
Order Readers";
@If(DeptCode = "EMEN";Proposer:CFO:"[CEO]":"[President]":"Purchase
```

```
Order Readers";  
""))
```

For POs submitted from most departments, reader access is controlled by the settings in the Form properties box. This formula further restricts reader access to POs from four departments.

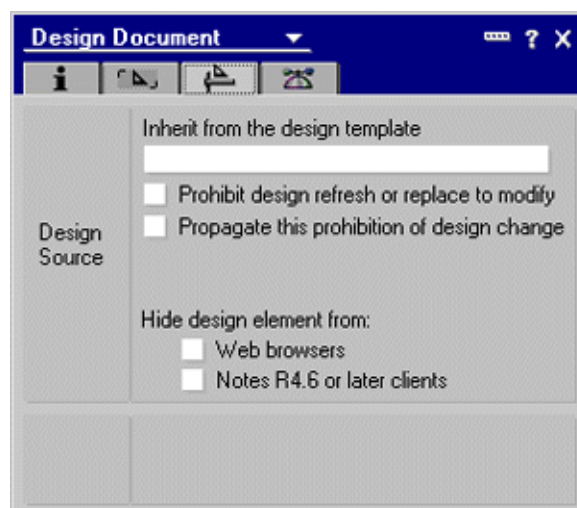
The surest way to hide data in a Domino application is to use a readers field in the documents that include the sensitive data. Only the persons or groups defined by the readers field will have access, and all other users will be denied. Users who do not have access to a document will not see it listed in views. If you want to make sure users are unaware of the categories of documents they cannot access, set the properties for each view in your application to "Don't show categories having zero documents." (It's on the Advanced tab of the View properties box.) This feature is new in Release 5.

Client-related hide-whens

Making an application's UI work in both Notes clients and Web browsers can require a great deal of hide-when work. UI features that work in one client may not work in the other. For instance, Notes supports tabs, but Web browsers don't. Attachments go in a rich text field in Notes, but they require an upload control in a Web browser.

If the application is simple enough, a single form can handle both clients. Release 5 of Domino has made it easier by adding an @function, @ClientType, to make it simple to evaluate the user's client platform. You can hide design elements by writing hide-when formulas that evaluate the client type.

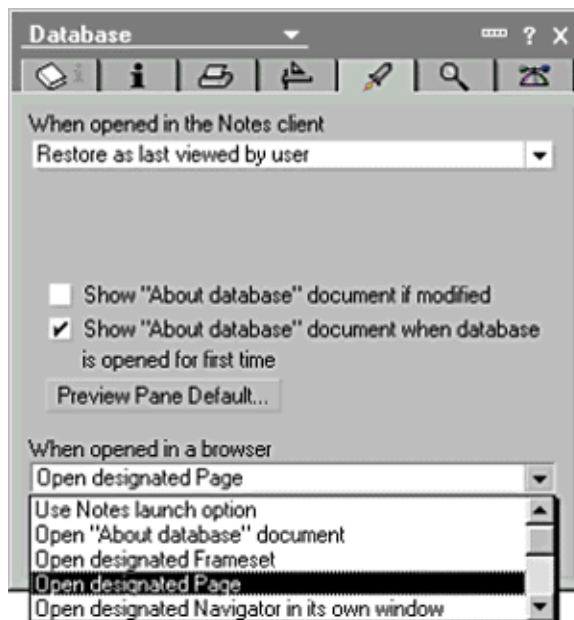
If you design elements specifically for one client or the other, you can enforce your choice by hiding the element from the client it isn't designed for in that element's design properties box. In Designer, right-click on the element's name and choose Design Properties. Then select the Design tab of the properties box (T-square and triangle) and under "Hide design element from" select either the Notes clients or Web browsers. This works for virtually all design elements—outlines, framesets, pages, forms, views, folders, navigators, and resources.



Additionally, scripted events can help adjust forms for compatibility with different clients. Release 5 provides new events, WebQueryOpen and WebQueryClose, to allow you to script additional processing of forms intended for browsers.

There comes a point, however, when the complexity of what's hidden and what's revealed makes it simpler to create separate forms for each client, or to use a computed subform to deliver the right UI to the right client. The simplest approach is usually to create two parallel paths through the application's architecture—one set of forms and views for Notes clients, another for Web browsers, both sharing the same documents.

The easiest way to start the user down the right path is to use the Launch options in the Database properties box. Launch the database on the Web by opening any one of the possible options—a frameset that contains a Web-friendly view, perhaps, or a page that serves as a home page for the application on the Web. Make your choice from the list that opens under "When opened in a browser" on the Launch tab:



There is one "gotcha" you need to keep in mind when hiding elements from particular clients. You can't hide fields from browsers if you want their values to be accessible to JavaScript or CGI scripts. It's common to pass a value in a pure HTML form while hiding it from users:

```
<INPUT type=HIDDEN name="metasearch" value="yes">
```

If you use a hide-when to hide a field from the Web in a Domino application, however, it is omitted from the HTML page generated by Domino.

Hide-when exceptions

Just about any UI element can be hidden in a Domino application. There are two exceptions, and they are the source of most problems with hide-whens in complex applications: rich text fields and tables.

The problem with both is that they can contain elements that can be independently hidden or revealed, so the magic of hide-whens can't work reliably on them. For each element, Domino makes compromises in order to "do the right thing."

No hocus-pocus for tables, but...

For tables, the compromise is to do nothing at all. A table can't be hidden. There is no Hide tab in its properties box. But Domino does the right thing for the contents of individual cells: it treats them as paragraphs. A field in a table cell can be hidden or revealed. Multiple paragraphs within a cell can be

individually hidden. Domino does the right thing with table rows, as well: when all the contents of a row are hidden, the row itself is hidden and the space it occupies is closed up. Because the right thing for table columns is harder to predict, the Domino compromise is to *not* close up the column space if all its contents are hidden.

The effect is that you *can* hide a table by hiding everything within it. But if a row of the table is visible, all its columns are visible, even though the contents of some cells may be hidden.

For rich text fields, keep it simple

Nowhere does the logic of "the right thing" for hide-whens become more twisted than in rich text fields. Unlike tables, you can hide rich text fields. But a hide-when for a rich text field is really more of a suggestion than an absolute imperative. The reason is that paragraphs within a rich text field can be given hide-when attributes different from the field as a whole. Notes applies these attributes from the outside in: if an element inside the field contradicts the properties of the field itself, the element wins and the field is displayed in accordance with the properties of the element.

This is all in the name of doing the right thing by the content. The displayed result may be right for the content but a surprise for the developer. It is possible to pretzel the logic of a rich text hide-when to the point that the element that is forcing a field to display is itself hidden.

Even if the contents of a rich text field can be expected to behave themselves, hiding the field can still be problematical. Using a hide-when depends on being able to evaluate a condition. The simplest conditions are no problem—is the form in Edit mode or Read mode? But anything more complicated usually depends on evaluating the contents of the field itself. The most common test is usually to see if the field has contents, and hide it if it doesn't:

```
ThisField = ""
```

This works fine for text fields, but not for rich text fields, which do not evaluate to empty. A common workaround for rich text fields that are guaranteed to contain text is to use a flag field (we'll call it FlagField) and the @Abstract function. Create a computed text field and give it the formula:

```
@Abstract([abbrev]; 200; ""; SomeRichTextField)
```

Then you can give the rich text field named SomeRichTextField the hide-when formula:

```
FlagField = ""
```

This works only if the document has been saved, since rich text fields are not part of a document until it has been saved. And @Abstract doesn't work, of course, if the contents of the rich text field are something other than text.

The best solution for these situations is to avoid them. Don't use rich text fields in an application unless you must, and don't apply hide-whens to the few that you use unless you can be sure that the contents of the field will always be what you expect them to be.

Put a space in front of it

There is one more workaround for problems with hiding rich text fields that needs wider distribution. If you can't get a rich text field to hide no matter what you do, put a space in front of it. [Lotus Customer Support Technote #158231](#) reports:

"When a rich text field is the first item on a line and the form is saved before

hide attributes are added to this field, hide attributes added to this field later will not work. If text, even a blank space, or another field is placed before the rich text field on the line, the field will then hide properly."

Sections are similar

The logic of "the right thing" in sections can seem illogical, too. You can hide a section, the same way you hide any other object on a form, but it does a sort of Cheshire Cat act—parts of it hide, but parts of it stay in plain site. Here's an example:

You create a section that contains a few fields. In the Hide tab of the Section properties box, you write a formula to hide the section based on the value of a field above. When this section's formula evaluates to true the section title will be hidden. But all the fields within the section will remain visible. Is there logic in this? Yes. The reason is that on a form fields are not actually contained within a section: they aren't child-objects of the section object, they are child-objects of the form itself. So while you might expect them to inherit the properties of the section, they don't. You have to set them all individually. Fortunately, you can do this in one shot—highlight the entire section (fields as well as the section title), click the Properties SmartIcon, and then add the same hide-when formula as the section has to the Hide tab of the Text properties box.

Other hide-when tips and techniques

Hide-whens seem to show more of their capricious personality when used in tables: The [Notes/Domino Gold Release Forum](#) regularly carries posts from developers seeking a solution to problems with wandering hide-whens that move from cell to cell, or a runaway hide-when formula that replicates itself in every cell. The connection with tables, however, is tenuous, because similar problems can occur even when there are no tables involved. Experience indicates that the problems are most likely resource-related: the more complex a form gets, and the longer the editing session goes on, the more likely problems are to occur. The best advice seems to be to make sure your PC has plenty of available memory and system resources.

If you're having trouble with hide-whens, whether in tables or not, try these tips, many of them gleaned from the Gold Release Forum:

- Create and position all the fields in the form before you attempt to set their hide-when properties. Adding new fields and text, cutting and pasting to rearrange a form, adding or deleting table rows—all these things change the relationships between design elements and paragraphs, which can change the hide-when properties of the elements involved.
- Make sure your problem isn't "pilot error." Make sure that elements that are supposed to have hide-when properties or formulas do indeed have them and that those that shouldn't, don't. If you add or move fields, recheck to make sure the proper formulas are still there. If you insert paragraphs or elements, make sure they haven't inherited hide-when attributes from "parent" paragraphs.
- If you create a formula in the "Hide paragraph if formula is true:" box, be sure the checkbox is also selected; the formula won't be recognized until the box is checked. If you uncheck the box, be sure to also delete the formula; a formula, once recognized, may continue to be recognized until it is removed, regardless of the state of the checkbox.
- Save the form frequently as you work on it. If you suspect that your machine is resource-bound (if your fonts occasionally vanish, or windows won't open or close, or applications lock up), conduct your Designer session on as clean a machine as possible. Shut down all nonessential windows and applications. It may help avoid problems to periodically shut everything down and restart your PC regularly within your editing session.

- As you work, make copies of the form so you can revert to a recent version if necessary, and when you finish, make a safe backup of the working form.
- In extreme cases, save after each formula change.
- If you are having trouble with hide-whens not working, compact the database. (Even if it doesn't work, it gives you a quiet minute or two of meditation to slow your heart rate and calm your breathing.) If the formulas appear to be correct but the form still is not working, try deleting formulas, saving the form, then recreating the formulas a few at a time.

Conclusion

Hide-whens are an important tool for creating an optimal user interface for a Domino application. They allow the developer to tailor a form to the needs of the user, whether reading or editing, and to the capabilities of the client. But hide-whens add another layer of complexity to forms that are often already complex. The results can push both Domino and the developer to their limits. Understanding those limits, and making sure the simple structures of the form work before adding complexity, are the key to a good results.