

Part 2 – Adding Functionality to Forms

Overview

Introduction This is the second in a four part series on the basics of developing databases with Domino Designer. In this part you will use formulas to add greater functionality to your database forms.

In this module This module contains the following lessons:

Lesson	See Page
Lesson 6: Formula Basics	81
Lesson 7: Using @Functions in formulas	88
Lesson 8: Entering formulas in fields	102
Lesson 9: Response Forms	109
Lesson 10: Form Actions	117
Lesson 11: Hide-When Options	130
Lesson 12: Enhancing Form Layout with Tables	136
Lesson 13: Retrieving Data Stored in Documents	141
Lesson 14: Incorporating Workflow	149

Lesson 6: Formula Basics

Overview

Introduction This lesson introduces Notes formula basics, including formula components and syntax.

Objectives At the end of this lesson you will be able to:

- Describe the components of a Notes formula

In this lesson... This lesson contains the following topics:

- Components of a Notes formula
- Formula syntax rules

Components of a Notes formula

Formulas and @functions

A Notes formula is an expression that has program-like attributes and follows a set of syntax rules. The expression can contain one or more statements, executed in order. Formulas can range from very simple to very complex. An @function is a special type of built-in Notes formula.

Where are they used?

Formulas can be used in numerous locations throughout an application including the following:

- Editable fields
 - Computed fields
 - Action buttons
 - Outlines
 - Agents
 - View columns
 - View selection
 - Hotspots
 - Actions
 - Window titles
-

Formula Components

Notes formulas can contain the following components:

- Variables and temporary variables
- Constants
- Operators
- Formula keywords
- @Functions and @Commands

The next several pages describe and provide examples for each of these components in detail.

Continued on next page

Components of a Notes formula, Continued

Variables A variable is a value that changes. You will commonly use field names in your formulas as variables. The content of the field will vary from document to document, so the value that the formula is operating on will change from document to document.

Example: The following examples assume a field exists with the name `FirstName`.
Variables

Use of field name in formulas	Example
Alone	<code>FirstName</code>
With operators and a constant to produce a new value	<code>FirstName + " " + LastName</code>
As an argument in an @Function	<code>@UpperCase(FirstName)</code>

Temporary Variables A temporary variable exists only within a formula. Its scope is the formula and it has no attributes other than the ones assigned to it within the formula. Any data assigned to a temporary is not saved. Temporary variables are useful in longer formulas, making the formula easier to interpret and debug.

Temporary variable syntax:
`tempname := value`

Example `firstname := "John";`
Temporary `lastname := "Smith";`
Variable `@ProperCase(firstname) + " " + @ProperCase(lastname)`

Continued on next page

Components of a Notes formula, Continued

Constants A constant is a value that does not change. Some constant values require special punctuation.

Examples: The following table describes the types of constants used in Notes formulas:
Constants

Type of Constant	Description	Example
Text	Any characters enclosed in quotation marks (“”).	“User One” “New York”
Number	Any number 0 to 9 Negative numbers use (-)	245 -900
Time/Date	A date or time must be enclosed in square brackets ([])	[01/01/99]

Operators Operators are used in formulas to modify, combine or compare values.

Examples: The following table lists and describes types of operators for Notes formulas:
Operators

Operator Type	Description	Symbol(s)	Example
Assignment	Used to assign a value to a variable, such as a field name	:= (colon, equal)	FirstName := “John”
Concatenation	Used to join pieces of text, fields or the elements in a list	text + list :	Street + “,” + City “Sales” : “HR” : “Training”
Arithmetic	Used for mathematical calculation with numeric constants or number fields	Addition + Subtraction – Multiplication * Division /	TotalPrice + Tax TotalPrice – Discount UnitPrice * 10 TotalPrice/Quantity
Comparison	Used to compare constant values or field contents	Equal = Not equal != Less than < Greater than >	Month = “January” Month != “March” Price < 100 Quantity > 10
Logical	Used to perform multiple evaluations within a formula	AND & OR (pipe symbol) NOT !	TotalPrice > 100 & Quantity < 10

Continued on next page

Components of a Notes formula, Continued

Formula Keywords The formula language contains a set of five keywords that perform special functions. The formula keyword is always the first word of a statement. By convention keywords are entered in uppercase.

Examples: The following table describes the five formula keywords:
Keywords

Keyword	Description	Example
FIELD	Used to assign a value to a field. If the field does not exist, Notes will create it.	FIELD Address := "818" + Street
SELECT	Defines a set of documents for a view, agent, or replication formula.	SELECT City = "New York"
REM	Indicates that a statement line contains comments only and should not be evaluated.	REM "The next line sets the contents of the City field"
DEFAULT	Will either assign a default value to a field or state that for the duration of the formula act as though the document has a field with this value.	DEFAULT City := "New York"
ENVIRONMENT	Sets an environment variable stored in the user's NOTES.INI file.	ENVIRONMENT Department := "HR"

@Functions and @Commands @Functions are built-in formulas that perform specialized calculations or operations. @Commands are a specific type of @function designed to mimic most of the Notes/Domino menu functionality. @Functions will be covered in detail in Lesson 7.

Formula Syntax Rules

Introduction Notes formulas must follow certain rules regarding syntax. Syntax refers to the general structure of the formula and the use of separators, case, operators and parentheses.

Separators Notes recognizes three separators. The following table describes them.

Separator	Purpose	Example
Colon :	Separates multiple items in a list.	“Austin” : “Paris” : “Rome” Street : City : State
Semicolon ;	Separates multiple statements in a formula or arguments of @functions.	SELECT City = “Paris”; FIELD Country := “FR”
Spaces	Any number of spaces can be used between formula elements. At least one space must follow a keyword.	REM “Comments Only”

Case The Notes formula language is not case sensitive except within quotation marks. For example, “new york” is different from “New York”. The following tables shows some of the conventions in Notes formulas regarding case.

Component	Convention	Example
Keywords	Upper case	FIELD, REM, SELECT
Field names	Proper case, each word	City AuthorName
@Functions	Proper case	@Created @UpperCase

Quotation marks Quotation marks should only be used to enter text constants. Field names should not be enclosed in quotation marks, although there are @functions that are exceptions to this rule, such as @SetField and @DbLookup.

Continued on next page

Formula Syntax Rules, Continued

Parentheses Parentheses are used to enclose the arguments of @functions and to control the order of operations in a mathematical statement. You must always use the same number of open and close parentheses.

Data types All data used in a formula must be of the same data type or an error will result. For example, in the following formula, if the ItemPrice field is a number data type the result will be an error:

“The price of this item is “ + Item Price

This formula is attempting to concatenate text and numbers. In order for the formula to work, the ItemPrice field must be treated as text for the purpose of this formula. The following formula will work correctly:

“The price of this item is “ + @Text(ItemPrice)

The data conversion @functions will be covered in detail in Lesson 7.

Logical operators When using the logical operators & (AND), | (OR) and ! (NOT), the expressions on either side of the operator must each have a Boolean (true or false) result that can be evaluated on its own, such as City = “Paris” & Country = “FR”

When using the logical operators in a formula, use the symbols rather than the words.

Lesson 7: Using @Functions in Formulas

Overview

Introduction In this lesson, you will be introduced to Notes @Functions. @Functions are a very important part of the design of most Domino applications.

Objectives At the end of this lesson you will be able to:

- List the categories of @Functions
 - Evaluate existing @Functions
-

In this lesson... This lesson contains the following topics:

- @Function Basics
 - String @function examples
 - Date/Time @function examples
 - Arithmetic @function examples
 - List @function examples
 - Logical @function examples
 - Data conversion @function examples
 - User Input @function examples
 - Miscellaneous @function examples
-

@Function Basics

Definition An @function is built-in formula designed to perform specified tasks within a Notes or web application. There are over 200 Notes @functions. When looking for a certain type of functionality in your design, check the available @functions to see if there is one to provide the desired performance.

Uses for @functions @Functions are used in Notes to perform numerous type of functionality including the following:

- Formatting text strings
 - Generating and formatting dates and times
 - Evaluating conditional statements
 - Calculating numeric values
-

Syntax Some @functions require arguments; other produce a value without any arguments. The general syntax of an @function is:

@Function with	Syntax	Example
No arguments	@FunctionName	@Today
A single argument	@FunctionName(arg)	@UpperCase(City)
Multiple arguments	@FunctionName(arg1; arg2; arg3)	@If(Total > 1000; "Over Budget"; "Within Budget")
Keyword arguments	@FunctionName([Keyword]; arg)	@Name([CN]; Author)

Types of @Functions Notes @functions can be divided into the following categories:

- String manipulation
 - Date/Time operations
 - Arithmetic operations
 - List operations
 - Logical evaluations
 - Data conversion
 - User input
-

For additional information All @functions are described in detail in the *Domino Designer 7 Help* database.

String @Function Examples

Introduction String @functions perform operations on text strings, generally the contents of a text field. String @functions:

- Work only on text fields or text constants entered in quotation marks.
 - Take variables or constants as arguments.
 - Require quotation marks around text constant arguments.
-

Examples: The following table lists and describes examples for commonly used string
String @functions:
Manipulation

@Function	Description	Example	Result
@ProperCase(string)	Provides initial capitalization for each word in a string.	@ProperCase("user one")	User One
@Trim(string)	Removes leading, trailing, and extra spaces from a string.	@Trim(" Los Angeles ")	Los Angeles
@Left(string; number)	Returns specified number of characters from the left of a string	@Left("New York"; 3)	New
@NewLine	Inserts a carriage return into a text string; used for text appearance	Street := "123 Main St"; City := "Dallas"; Street + @NewLine + City	123 Main St Dallas
@Contains(string; substring)	Determines whether a substring is stored anywhere within a string.	City := "Los Angeles"; @Contains(City; "Los")	1 (true)

Date/Time @Function Examples

Introduction Date/Time @functions are used to generate or manipulate date and/or time values. A date/time function will only work correctly on date/time values; any other type of data will generate an error message.

Examples: The following table lists and describes examples for commonly used Date/Time Operations date/time @functions:

@Function	Description	Example	Result
@Created	Displays the creation date of a document; the creation date is automatically stored internally	@Created	The date that the document was created.
@Adjust(date; years; months; days; hours; minutes; seconds)	Adjusts the date by the specified increments, use the – sign to adjust the date backwards	@Adjust([01/03/99]; 2; 0; 20; 0; 0; 0) (Date = MM/DD/YY)	01/23/2001
@Today	Returns the current date.	@Today	Today's date
@Now	Returns the current time and date	@Now	Current date and time, including seconds.
@Month(date)	Returns the number of the month from the specified date	@Month([12/25/2001])	12

A note about @Adjust The @Adjust function can produce unexpected results if you don't understand the order in which the date is incremented. While the arguments are given in order from the largest to the smallest, when the function is evaluated, the units are added to the date in order from smallest to largest.

Example: @Adjust([02/28/98]; 2; 0; 1; 0; 0; 0)

First, 1 day is added, making the date 03/01/98. Then 2 years are added, making the date 03/01/2000.

Arithmetic @Function Examples

Introduction Arithmetic @functions perform calculations on numeric values.

Examples: The following table describes and lists examples of commonly used arithmetic @functions:
Arithmetic Operations

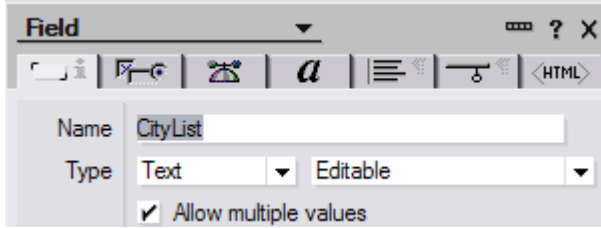
@Function	Description	Example	Result
@Max(number; number)	Returns the larger of two numbers or the largest value in a list.	Total := 10; @Max(Total; 30)	30
@Min(number; number)	Returns the smaller of two numbers or the smallest value in a list.	Total := 10; @Min(Total; 30)	10
@Round(number)	Rounds a number to the nearest whole number	@Round(100.9)	101

Caution for arithmetic @functions

A blank or empty number field does not have a value of zero; it is considered a null or empty string. If you try to use an arithmetic @function on an empty number field, it will produce an error. The number field must have a default value or you must test for the contents of the number field as part of the formula.

List @Functions Examples

Introduction List @functions evaluate and manipulate lists. Some @functions that operate on lists also operate on single text items as well. List @functions are frequently used on fields that use the property “Allow multiple values”.



Examples: The following table lists and describes examples for list @functions:
List operations

@Function	Description	Example	Result
@Elements(list)	Counts the number of items or elements in a list	@Elements(“user one” : “user two”)	2
@Member(value; list)	Returns the number that corresponds to the position of the value in the list. Returns 0 if the value is not in the list.	Dept := “Sales” : “HR” : “Training”; @Member(“HR”; Dept)	2
@Subset(list; number)	Returns the number specified from the list, counting from the left. If the number specified is negative, it will count from the right.	Cities := “Rome” : “Paris” : “Chicago”; @Subset(Cities; 2)	Rome; Paris
@Explode(string) or @Explode(string ; separator)	Converts a text string into a text list.	Cities := “Rome Paris Berlin”; @Explode(Cities)	Rome; Paris; Berlin
@Implode(textlist) or @Implode(textlist ; separator)	Concatenates all members of a text list and returns a text string.	Cities := “Rome” : “Paris” : “Berlin”; @Implode(Cities; “, “)	Rome, Paris, Berlin
@Sort(list;[ORDER])	Used to specify the order in which elements of a list are displayed.	Cities := “Paris” : “Rome” : “Berlin”; @Sort(Cities; [ASCENDING])	Berlin, Paris, Rome

Logical @Function Examples

Introduction Logical @functions produce values based on the evaluation of a condition. The most common logical function is @If, which is used throughout Notes applications. Additional logical functions include:

- @For
 - @While
 - @DoWhile
-

Examples: The basic syntax of the @If function is:
@If @If(Condition; true-action; false-action)

The @If can also evaluate multiple conditions without the need to nest a second @If statement. A single @If statement can evaluate up to 99 conditions.

@If(Condition1; Action1; Condition2; Action2; False-Action)

The following table lists and describes examples of the @If function:

@If example	Result
@If(DueDate<@Today; "Past Due"; "On Schedule")	If today's date is greater than the DueDate the text "Past Due" will be returned. Otherwise "On Schedule" will be returned.
@If(Department = "Sales"; "Building H6"; Department = "Training"; "Building H8"; "Building H10")	If the Department field equals "Sales" then "Building H6" is returned. If the Department field equals "Training" then "Building H8" is returned. If neither condition is true then "Building H10" is returned.

Guidelines for @If @If statements:

- Always have an odd number of arguments
- Take up to 99 conditions
- Can be nested
- Are commonly used with other @Functions

Continued on next page

Logical @Function Examples, Continued

Using @If for field validation

Frequently, the designer will want to verify that the contents of a given field meet certain criteria. This is called field input validation and the @If statement is commonly used. Using an @If statement to validate field contents prevents the form from being saved until the field passes the validation formula.

Example 1: Field validation

The following statement verifies that the Department field is not left blank.

```
@If(Department = NULL; @Failure("You must select your department."); @Success)
```

When testing for an empty field, you can use either NULL or "".

Example 2: Field validation

The following statement verifies that the Quantity field contains a value of 1 or greater.

```
@If(Quantity => 1; @Success; @Failure("You must enter a quantity of 1 or greater.))
```

Using @If to check for null values

Because a blank number field is considered a null string and not a value of zero, if you attempt to use the field in an arithmetic formula or function you will receive an error. To avoid the error, you can enter a default value in the number field or you can test for a null string as part of the arithmetic formula.

Example: Testing for a null string

The following example could be used in a field to calculate Price * Quantity:

```
@If(Price = NULL | Quantity = NULL; NULL; Price * Quantity)
```

If either the Price field or the Quantity field is blank, the formula will not perform the multiplication and will return a NULL value to the field where this formula is evaluated.

Continued on next page

Logical @Function Examples, Continued

@For The @For function executes one or more statements iteratively while a specified condition remains true. @For executes the initialize statement once, then evaluates the condition. If the condition is True, @For executes the statements, executes the increment statement, and evaluates the condition again. If the condition is False, @For terminates.

@For syntax The syntax of the @For function is as follows:

@For(initialize ; condition ; increment ; statement ; ...)

The following table lists and describes the @For parameters:

Parameter	Description
Initialize	A statement that assigns an initial value to a variable in the <i>condition</i> parameter.
Condition	Expression that returns a true or false value.
Increment	A statement that increments the initialized variable. The increment parameter is very important. If no increment occurs in the formula, the formula will be caught in an endless loop and Notes will have to be shut down and restarted.
Statement	A formula language statement. The maximum number of statements you can include is 254.

@For example This code, when used in the FullName field, concatenates the list elements in the FirstName and LastName fields:

```
@For(n :=1; n<=@Elements(FirstName); n := n + 1;  
FIELD FullName := @If(n=1;FirstName[n] + " " + LastName[n]; FullName :  
(FirstName[n] + " " + LastName[n])); FullName
```

If FirstName contains:

"John" : "Patty" : "Mary"

and LastName contains:

"Williams" : "Keats" : "Foster"

the result in the FullName field is:

"John Williams; Patty Keats; Mary Foster"

Continued on next page

Logical @Function Examples, Continued

@While The @While function executes one or more statements iteratively as long as a specified condition remains true. @While first evaluates the condition. If the condition is true, @While executes the statements then evaluates the condition again. If the condition is false, @While terminates. If the condition starts out false, the statements will never be executed.

@While syntax The syntax of the @While function is as follows:

@While(*condition* ; *statement* ; ...)

The following table lists and describes the parameters of @While:

Parameter	Description
Condition	Expression that returns a value of True or False.
Statement	A formula language statement. The maximum number of statements you can include is 254.

@While example

This action displays the elements of the Categories field one at a time.

```
n := 1;
```

```
@While(n <= @Elements(Categories);
```

```
@Prompt([OK]; "Category " + @Text(n); Categories[n]);
```

```
n := n + 1)
```

Continued on next page

Logical @Function Examples, Continued

@DoWhile The @DoWhile executes one or more statements iteratively while a condition remains true. @DoWhile executes the statement(s) then evaluates the condition. If the condition is true, @DoWhile executes the statement(s) and evaluates the condition again. If the condition is false, @DoWhile terminates. The statements will always be executed at least once, because they are executed before the condition is tested.

If you are looping through a field containing a list, be sure the Allow multiple values check box is selected in the Field Properties box for the list field.

Syntax The syntax for the @DoWhile is as follows:

```
@DoWhile( statement ; ... ; condition )
```

The following table lists and describes the parameters of the @DoWhile function:

Part	Function
Statement	A formula language statement. The maximum number of statements you can include is 254.
Condition	Expression that returns a value of True or False.

Example This action displays the elements of the Categories field one at a time.

```
n := 1;  
@DoWhile(@Prompt([OK]; "Category " + @Text(n); Categories[n]);  
n := n + 1;  
n <= @Elements(Categories))
```

Data Conversion @Function Examples

Introduction As mentioned earlier in this lesson, all the data used in a @function must be of the same data type or the result of the function will be an error.

Error Messages Below are the common error messages received if the data types are not correct:

“Field: ‘Total’: Incorrect data type for operator or @Function: Number expected”

“Field: ‘Department’: Incorrect data type for operator or @Function: Text expected:

Examples: The following table lists and describes examples of the data conversion @functions:
Data conversion

@Function	Description	Example
@Text(value)	Converts a number or date/time field into text for the current formula	@Text(RetailPrice)
@TextToNumber	Converts a text string into a numeric values for the current formula. The text string must be in numerals not words.	@TextToNumber(“100”)
@TextToTime	Converts a text string into a numeric values for the current formula. The text string must be in numerals not words.	@TextToTime(“1/1/99”)

User Input @Function Examples

Introduction

There are several @functions that provide a user input mechanism, rather than the standard field options. The information that the user enters is captured as the result of the @function and can be used in formulas or returned to and stored in fields.

Examples User Input

The following table lists and describes user input @functions:

Function	Description
@Prompt	Displays a dialog box to the user and returns a text value based on the user's actions in the dialog box. @Prompt is useful for prompting a user for information and determining a course of action based on the user's input.
@DialogBox	Brings up a dialog box that displays the current document. The dialog box shares fields with the underlying document. The user interacts with the dialog box as usual, clicking OK or Cancel when finished.
@PickList	Displays a modal window that contains either: <ul style="list-style-type: none">• A view you specify from which the user can select one or more documents. @PickList returns a column value from the selected document(s).• The Address dialog box, displaying information from all available Name & Address books. The user can select one or more person, group, server, room, or resource names, and @PickList returns those names.

Miscellaneous @Function Examples

Introduction There are several functions that don't fit into any of the previous categories. These @functions perform special functionality.

Examples: The table below describes and gives examples for commonly used @Functions:

Function	Description	Example
@UserName	Returns the name of the current user from the active ID file in canonical format.	@UserName
@IsNewDoc	Determines if the active document has been saved previously	@IsNewDoc
@Name([action];name)	Manipulates a hierarchical name. The action specifies what will be done to the name.	@Name([CN]; @UserName) Returns the common name of the current user ID file.
@Failure(string message)	Returns a message to the user in a dialog box. The designer supplies the exact text of the message.	@Failure("You have entered an invalid entry in the Start Date field")
@Success	Returns 1 (True). Use this function with @If in field validation formulas to indicate that the value entered satisfies the validation criteria.	@If(Department = NULL; @Failure("You must select a department"); @Success)

Lesson 8: Entering Formulas in Fields

Overview

Introduction Now that you are familiar with formulas and functions, you will want to add several into the Time-Off Request form. This lesson will show you the different types of field formulas and how to work in the Programmer's Pane.

Objectives At the end of this lesson you will be able to:

- Enter default value formulas in an editable field.
- Enter input translation formulas in an editable field.
- Enter input validation formulas in an editable field.
- Determine the appropriate type of computed field.
- Enter the formulas for computed fields.

In this lesson... This lesson contains the following topics:

- Editable field formulas
- Computed field formulas
- Working in the Programmer's pane

Editable Field Formulas

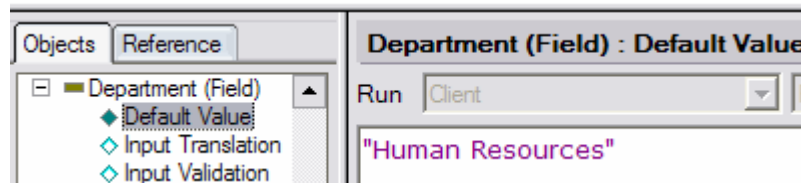
Editable field formula types

Editable fields accept user input, but editable fields can also use formulas. Editable fields can use *any or all* of the following types of formulas:

- Default Value
 - Input Translation
 - Input Validation
-

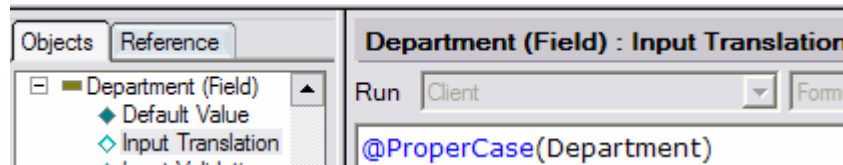
Default value

A Default Value formula sets a value in the field as the document is created. The default value can be modified by the user.



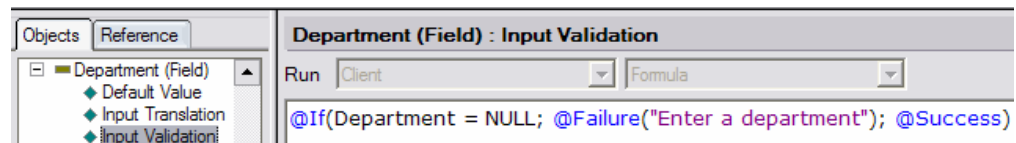
Input translation

The input translation formula modifies field contents. For example, an input translation formula can format user input to make all entries proper case. Input translation formulas are evaluated each time the document is refreshed, recalculated or saved.



Input validation

An input validation formula verifies that the contents of a field meet the criteria defined in the formula. For example, an input validation formula can ensure that users enter data into a required field. Input validation formulas are evaluated each time a document is refreshed, recalculated or saved.



Computed Field Formulas

Computed Fields

When the field type is changed from editable to computed, it no longer accepts input directly from the user. It must be populated by a formula. If you specify a field as computed and neglect to enter a formula for it, you will receive an error when you try to save the form. Each computed field must be *one* of the following types:

Field type	The field will be computed...
Computed	each time a user creates, saves or refreshes a document. Use this type of field when the value of the field will need to be updated.
Computed for display	each time a user opens, refreshes or saves a document. The field value exists while the document is on screen; it is not stored and cannot be used in a view. Use this type of formula to display information that is relevant only to the immediate session.
Computed when composed	only once, when the user first creates the document. Use this type of formula in a field to preserve information about the origin of a document, such as the creation date or original author. Rich Text fields cannot be computed when composed.

Order of form calculation

Computed fields are evaluated from the top down and left-to-right. Place dependent fields after the fields they depend on.

Selecting the computed type

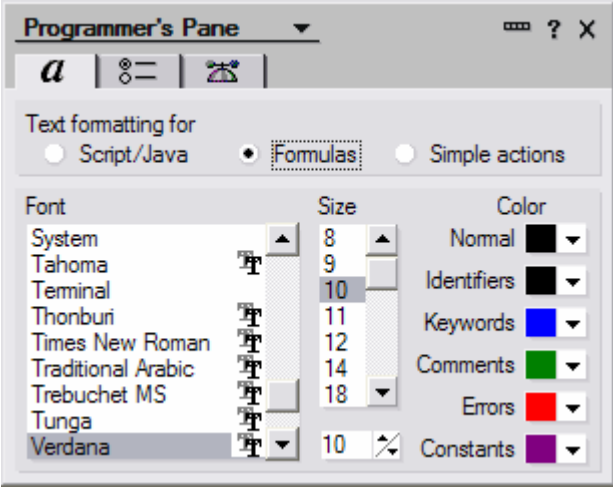
To change from editable to computed, complete the following steps:

Step	Action
1	Access the field properties.
2	Click the down arrow for type. <div data-bbox="695 1444 1295 1766" data-label="Image"> <p>The screenshot shows a 'Field' dialog box with a toolbar at the top. The 'Name' field contains 'RequestedBy'. The 'Type' dropdown menu is open, showing four options: 'Editable' (highlighted), 'Computed', 'Computed for display', and 'Computed when composed'. There are also checkboxes for 'Allow multiple' and 'Compute After', and a 'Style' section with radio buttons for 'Notes style' and 'Native US style'.</p> </div>
3	Select the appropriate type of computed field.

Working in the Programmer's Pane

Introduction The Programmer's Pane is the place where all formulas are entered in the Notes application. When a design element is opened, the Programmer's Pane is automatically displayed at the bottom of the screen.

Program Pane Properties The Programmer's Pane has several properties that can be changed to assist in creating in editing formulas. The properties that are set in the Programmer's Pane apply to the entire Designer client; every design element of every database will use the same properties. To access the Programmer's Pane complete the following steps:


Step	Action
1	Open a design element, such as a form.
2	Position the mouse pointer in the Programmer's Pane.
3	Right-click and choose Programmer's Pane Properties...
4	For properties that affect Notes formulas, select the Formulas radio button.
	
5	Select the desired font, size and color.
6	Close the properties box.

Continued on next page

Working in the Programmer's Pane, Continued

Entering the formula


To enter a formula for a field, editable or computed, complete the following steps:

Step	Action
1	Click on the field that requires the formula.
2	Click into the Programmer's Pane. If entering a formula for an editable field, select the formula type. For a computed field, the only option is Value.
5	Type the formula in the Programmer's Pane. If you start to type a function that is recognize by Designer, a pop-up menu will allow you to select the function from the list. <div data-bbox="730 814 1263 1352" data-label="Image"> </div>
6	After completing the formula, click the green check -  - to enter the formula. The red X will cancel the changes you made in the Programmer's pane. <div data-bbox="565 1486 1425 1644" data-label="Image"> </div>

Continued on next page

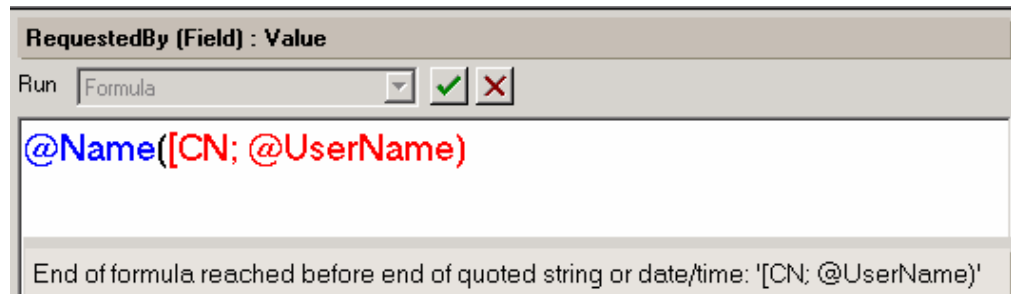
Working in the Programmer's Pane, Continued

Syntax check

When you type a formula and click the green check -  - to enter it, Notes performs an automatic syntax check. If a formula or an @function is entered incorrectly, an error message will be displayed at the bottom of the Programmer's Pane and point of the error will be highlighted in red, by default. The color of errors can be changed in the Programmer's Pane property box.

If a formula passes the syntax check, that does not necessarily mean that the formula is correct or that it will work as expected. For example, if a formula references a field name, but the field name is spelled incorrectly, the syntax check will not mark the misspelling as an error.

Example: Syntax check



The formula is missing a closing] after CN. The formula must be corrected before it can be saved.

Exercise 8

Entering Field Formulas on the Time Off Requests Form

Introduction In this exercise you will create formulas in existing fields of the Time-Off Request form.

Requirements & Resources This exercise will be completed in the Time-Off Request database using the Time-Off Request form. Some of the fields will remain editable, while others will need to be changed to computed fields. You will have to decide, given the desired outcome, whether the field will be editable or computed.

Instructions Enter formulas according to the following requirements:

1. When a new document is created, the RequestedBy field should automatically compute to the current user's first and last name only. The contents of this field should not change once the document has been created.
2. When a new document is created, the CreationDate field (this is a shared field) should automatically calculate the current date. The contents of this field should not change once the document has been created.
3. Every new document should start with a Request Status of "Pending".
4. The following required fields should be validated to ensure that they are not left blank. The user should be prompted with a message if any one of the fields is left empty.
 - DepartmentName
 - LeaveType
 - StartDate
 - EndDate

Checking your results The Time-Off Requests Part 2 database has examples of completed formulas for this exercise.

Lesson 9: Response Forms

Overview

Introduction At this point you have worked only with a Document type form. In addition to the Document type for forms, there are also Response and Response to Response type forms. In this lesson you will create and work with Response forms.

Objectives At the end of this lesson you will be able to:

- Create a Response type form
- Enable a Response form for inheritance
- Create formulas to inherit information from a Document into a Response document

In this lesson... This lesson contains the following topics:

- What is a Response Type Form?
- Creating a Response form
- Setting up inheritance in the Response form
- Inheritance Example

What is a Response Form?

Definition

A Response form provides all the same basic functionality that a Document form provides; it can contain fields, formulas, tables, graphics and more. Additionally a Response form provides a way for users to make comment on existing documents without modifying the originals. In most databases, users will not have the access to edit another user's document, so the Response form provides a way to add related information to the existing document. Documents and Responses are also referred to as Parent and Child documents.

Document vs. Response

If a database contains only one form, it must be a document type form. The Document type form can be used to create a new document independently from all other documents within the database. In contrast, a Response type form can only be used to create a document in response to an existing document.

If you attempt to create a new response document you must have an existing document open on the screen or selected in a view. The Response type form is dependent on other documents. If no document is open or selected, you will receive the following error message:



Response to response

In addition to the Response type form, there is a Response to Response type form. The Response to Response form is intended to allow users to create comments to a Response, but it can also be used to respond to a Document.

Continued on next page

What is a Response Form?, Continued

Viewing Response Documents

Generally, if a database has response documents the views will display them under the main documents, slightly indented.

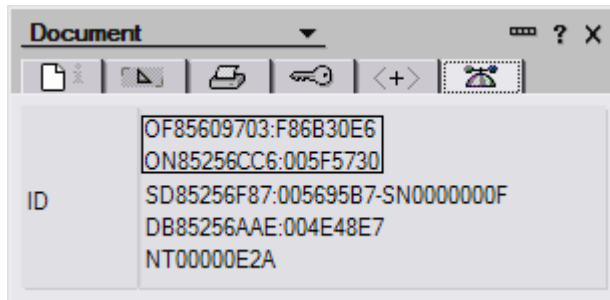
Department	Requestor Name	Type of Leave
▼ Approved		
▼ Training	User One	Vaction/Annual Leave

Example: Display of response document

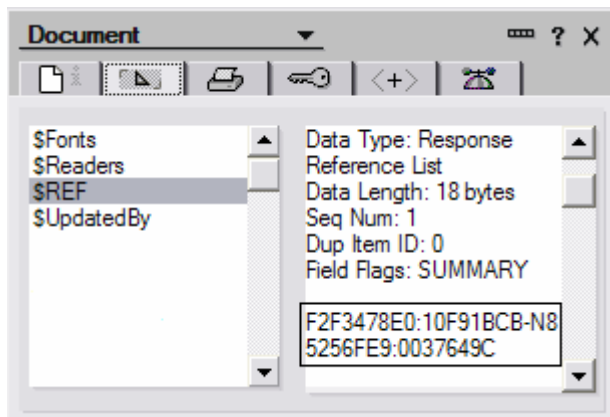
Modifying views to display responses in this way is covered in Lesson 17.

Document ID numbers and \$REF

Each document created in Notes stores an ID number. Part of that number is called the Unique ID, also referred to as UNID. The UNID of a parent document is outlined following graphic:




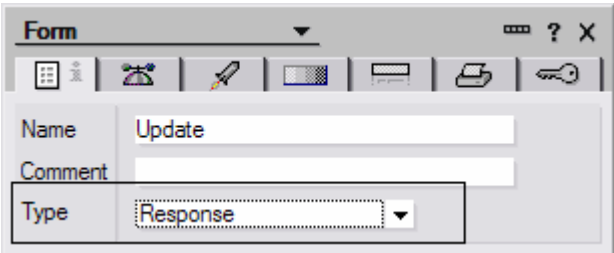
When a Response document is saved, it automatically stores the Unique ID number of the parent document in a field called \$REF.



Creating a Response Type Form

Procedure

To create a Response type form, complete the following steps:

Step	Action
1	In the Designer client, select Forms from the Design pane.
2	Click the button to create a new form - 
3	Right-click in the new form to access form properties.
4	In the properties box, click the down arrow for Type.
5	Select Response from the drop-down list. 
6	Complete the properties box as you would for any form.

Setting Up Response Form Inheritance

Introduction By default, when a user creates a response document, the new document does *not* inherit information from the main document. It can be very helpful, though, to set up inheritance so that the new response document automatically stores information about the main document to which the user is responding. For example, you might want the response document to store the title or subject and author of the main document. This can be done through a process called inheritance.

Steps to enable inheritance The following are the steps to enable inheritance on a Response form:

- Select the property “Formulas inherit values from selected documents” on the Defaults tab of the form property box.
- Determine the which fields from the main document the response document should inherit.
- Enter formulas in the fields of the response document to reference the fields that will be inherited from the main document.

Inheritance happens once The process of inheritance occurs only once, when the response document is first created. At the time of time of creation the formulas in the inheriting fields of the response document pull the information into the response document. This process does not happen again. Therefore, it is recommended that the inheriting fields be of type Computed when composed. If the inheriting fields are set as Computed, when the form is refreshed or saved the fields will re-evaluate the formula. The formula cannot inherit from the main document a second time, so the field would be left blank.

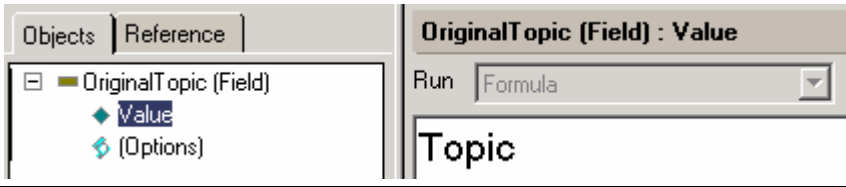
Field names in the response The names of the inheriting fields in the Response type form do not need to match the field names from the Document type form, but using the same field names may make it easier to recognize which fields are involved in the inheritance process.

Inheritance Example

Example;
Designing
inheritance

The following example describes how a designer sets up inheritance:

Stage	Description
1	The designer has created Document type form named Main Topic.
2	The Main Topic form has an editable text field named Topic.
3	The designer has created a Response type form named Response.
4	The Response form has the “Formulas inherit values from selected document” enabled on the form properties box.
5	The Response form has a field named OriginalTopic.
6	The OriginalTopic field is “Computed when composed”.
7	The OriginalTopic field has a formula that references the Topic field from the Main Topic form.



Example:
Using
inheritance

The following example describes how inheritance works from the user’s point of view:

Stage	Description
1	User One creates a document with the Main Topic form.
2	User One enters a topic of “Troubleshooting Modem Setup” in the Topic field.
3	User One completes the rest of the document and saves it.
4	User Two reads the document and wishes to make comments.
5	User Two has the document selected in a view and selects from the menu Create -> Response .
6	As the new Response document is created, the field OriginalTopic is automatically populated with “Troubleshooting Modem Setup”.
7	User Two completes the rest of the Response form and saves it.

Exercise 9

Creating a Response Form & Enabling Inheritance

Introduction Once the original request has been approved or denied we do not want the users to edit the requests. In this exercise you will create a new response form to allow users to create an update to their original time off requests.

Instructions Complete the following steps in the Time-Off Request database:

1. Create a response form named **Request Update** with an alias Update.
2. Enable inheritance on the form.
3. Create the text and fields listed in the table below. Use the information in the 3rd column of the table to determine field properties.
4. Save and test the response form.
5. Create and save at least 3 Request Update documents.

Suggested Static Text	Field Names	This field should...
Time Off Request for	OriginalRequestor	inherit the name of the requestor from the original request document.
Original dates for request	OriginalStart and OriginalEnd	inherit the start date and end date from the original request.
Original type of leave request	OriginalLeaveType	inherit the type of leave requested in the original request
Status of the original request	OriginalRequestStatus	inherit the status of the original request
New Request Dates	StartDateUpdate and EndDateUpdate	allow the user to enter a new start date and a new end date
New Leave Type	LeaveTypeUpdate	allow the user to enter a leave type
Status for update	StatusUpdate	allow the user to select from the choices of Open, Approved or Denied
Update created on	CreationDate (this should already exist as a shared field)	compute the creation of the Request Update

Continued on next page

Creating a Response Form & Enabling Inheritance, Continued

Results

The following fields should be populated with data from the original request:

- OriginalRequestor
- OriginalStart
- OriginalEnd
- OriginalLeaveType
- OriginalRequestStatus

The Request Update form should resemble the following:

Update to Time-Off Request for

Original dates for leave request: to

Original leave type of leave request:

The status of the original request is

New dates for leave: to

New leave type:

Status for the updated request: StatusUpdate

Update requested on

Lesson 10: Form Actions

Overview

Introduction Databases can make use of actions to provide the user an easy and accessible way to perform common menu tasks, such as saving, creating and printing documents.

Objectives At the end of this lesson you will be able to:

- Create an action button on the action bar of a form
- Set properties for action buttons and action bars
- Program an action with a pre-defined simple action
- Program an action with an @Command

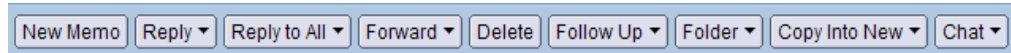
In this lesson... This lesson contains the following topics:

- What is an Action?
- Creating an Action
- Using Simple Actions
- Using @Commands in Actions
- Creating a Shared Action
- Inserting a Shared Action
- Cascading Actions
- Action Button and Action Bar Properties

What is an Action?

Introduction

An action is a custom procedure that you can associate with a view or form. When you open the view or open a document based on the form, the action becomes available as a menu command under Actions and/or as a button on an action bar. An action bar displays a collection of action buttons at the top of a form or view. A commonly accessed action bar is part of the mail template.



In this lesson we will look at adding actions to forms. The process is the same for adding actions to views or pages.

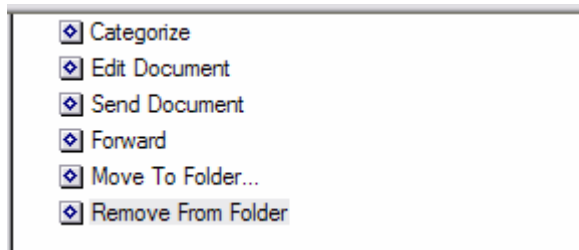
Programming Actions

Actions can be programmed by the following:

- Simple actions
 - Notes formula language
 - LotusScript
 - JavaScript
-


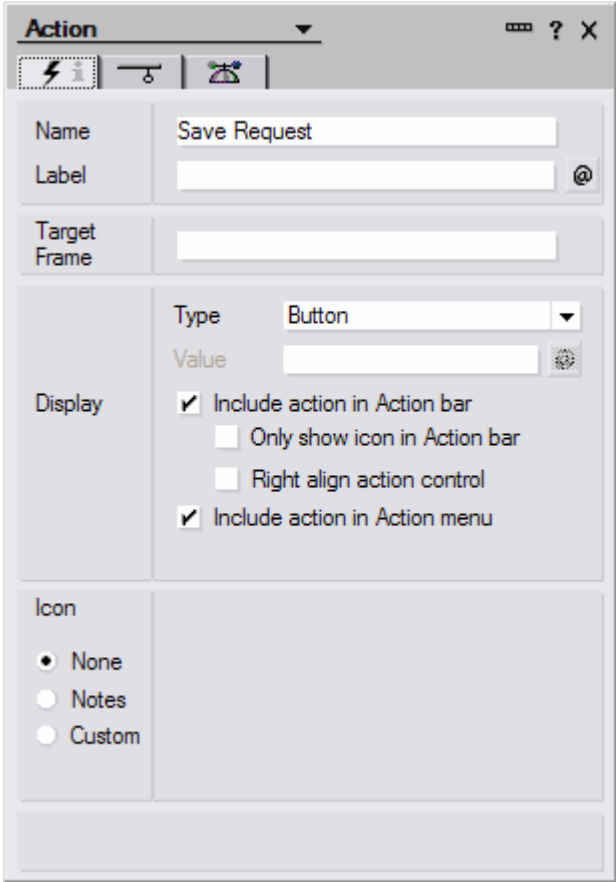
System actions

Each form and view has several system actions associated with it by default. They are not displayed automatically in the action pane, but you can add them by selecting Create -> Action -> Insert System Actions. Once added they will appear in the action pane.



Creating an Action

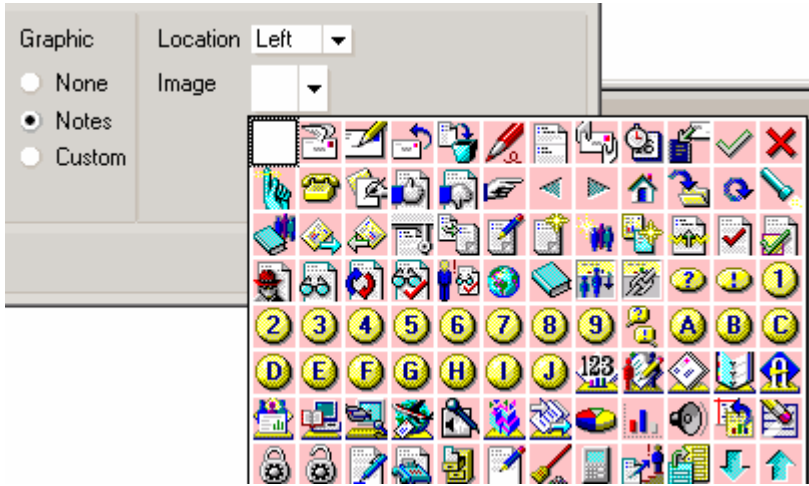
Procedure To create a form action complete the following steps:

Step	Action
1	Open the form where you want to create the action.
2	Open the action pane by choosing View -> Action Pane from the menu or clicking on the Display Action Pane icon –  The action pane will be displayed to the right of the work pane.
3	From the menu choose Create -> Action .
4	In the Action Properties box, type in a name for the action. 

Continued on next page

Creating an Action, Continued

Procedure (continued)

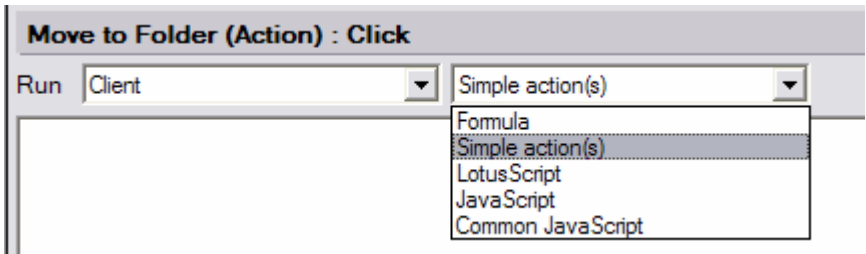
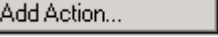
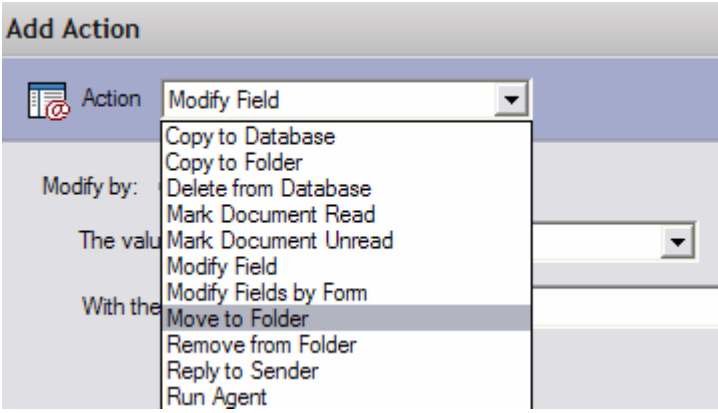
Step	Action
5	(Optional) Change the position of the new action in the Actions menu.
6	Select the desired display options, the action button bar and/or the Actions menu.
7	<p>Select a graphic to appear on the action button. Notes provides many graphic choices.</p>  <p>Selecting Custom will prompt you for an image resource from the database.</p>
8	Click in the Programmer's Pane to enter a formula for the action.

Note: The action bar does not appear in the Designer. You must preview the form in the Notes client or a web browser.

Using a Simple Action

Introduction Notes provides several simple actions that can be associated with an action button. These Simple Actions require no programming by the designer.

Procedure: To use a pre-defined simple action for an action button, complete the Simple Actions following steps:

Step	Action
1	Create a new action or select an action from the Action Pane.
2	Click in the Programmer's Pane.
3	Click on the down arrow for the Run option and choose Simple Action. By default, the Run option will be set to Formula. 
4	Click on the Add Action button at the bottom of the Programmer's Pane - 
5	Select the action from the list and click OK. 
6	Repeat steps 4 and 5 if the action button should perform multiple simple actions.

Using @Commands in Actions

Introduction An @Command is a special type of @Function that executes a Notes command. Most of the standard menu commands can be executed using @Command. In addition, a number of specialized commands are available.

@Command Syntax The basic syntax of the @Command is:
@Command([CommandName]; parameters)
Not all @Commands require parameters. For example, @Command([FileSave]) and @Command([FilePrint])

Reference For a complete list of all @Commands and the restrictions that apply to their use, see the *Domino Designer 7 Help* database, Formula Language @Commands A-Z.

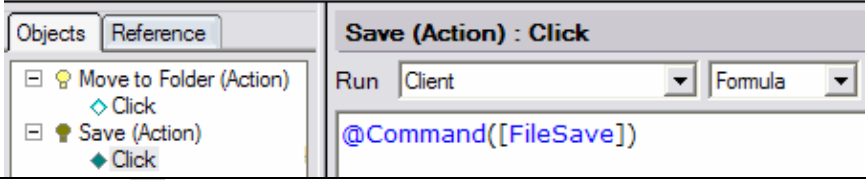

Examples The following table lists and describes some commonly used @functions for form actions:

@Command	This function
@Command([FileSave])	executes the menu command File -> Save to save the current document.
@Command([FileCloseWindow])	closes the current Notes window. If the document or design element in that window has not been saved, Notes prompts the user to save it before closing.
@Command([ToolsSpellCheck])	starts the Notes spell checker. The document must be in edit mode.
@Command([Compose]; "FormName")	creates a new, blank document using the form you specify. Optionally, you can specify the database, as well.
@Command([MailForward])	forwards the current document by placing its contents into a mail memo, which the user then addresses and sends like any other mail memo.
@Command([ToolsRefreshAllDocs])	Refreshes the fields of all the documents in a view or folder.

Continued on next page

Using @Commands in Actions, Continued

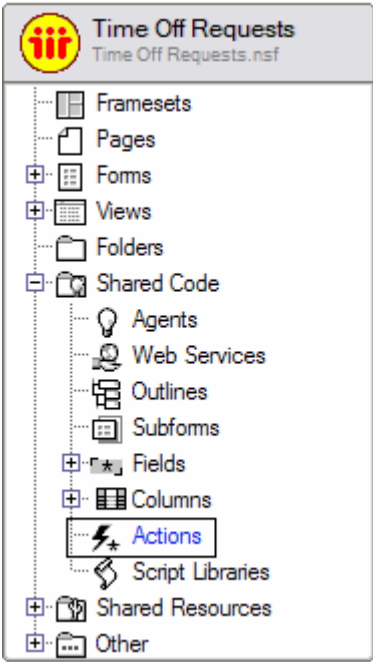
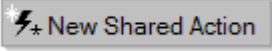
Procedure To enter an @Command in an action, complete the following steps:

Step	Action
1	Select the action from the action pane.
2	Click in the Programmer's Pane for the action.
3	Select formula as the run option for the action. Formula is the default run option.
4	Enter the @Command. To pick from a list of @Commands click on the Reference tab and select Formula @Commands from the drop-down list. 
5	Click the  button to accept the @Command.

Creating a Shared Action

Introduction You can also create shared actions in a database that can then be inserted into any form or view in that database.

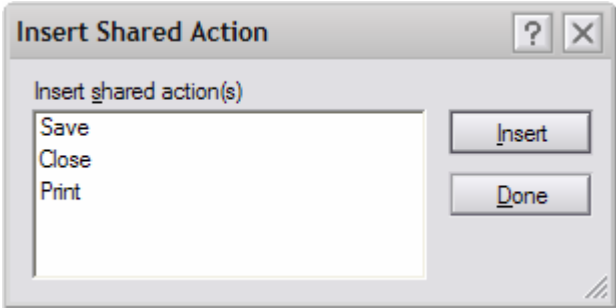
Procedure To create a shared action, complete the following steps:

Step	Action
1	From the Design pane, expand Share Code and click on Actions. 
3	Click on the button 
4	Enter information into the properties box, such as action name and display, as you would for a single-use action.
5	Enter a formula in the Programmer's Pane as you would for a single-use action.
6	Save the action and close the window.

Inserting a Shared Action

Procedure

Once a shared action has been created it can be inserted into a form or view. To insert a shared action into a form complete the following steps:

Step	Action
1	Open the Action Pane in the form where you want to insert the shared action.
2	Click on one of the existing actions in the Action Pane.
3	From the menu, choose Create -> Action -> Insert shared action.
4	Select the action and click Insert. 
5	The dialog box will remain open so you can insert multiple actions. Click Done when finished.

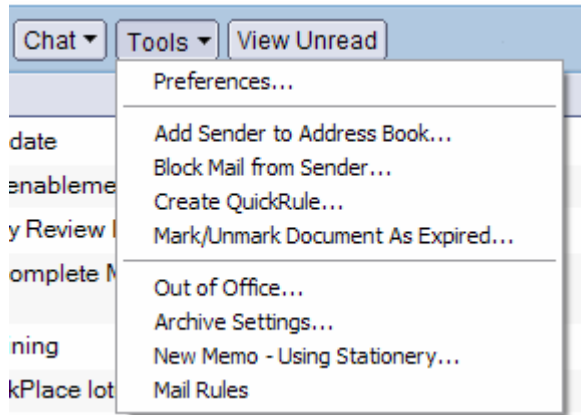
Modifying a Shared Action

A shared action cannot be modified from the action pane of a form or view. You must access it through **Share Code -> Actions**. Any changes that you make to a shared action will be reflected wherever that shared action is inserted.

Actions with sub-actions

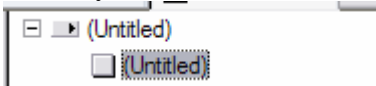
Introduction

If a form requires a large number of actions to be placed on the action bar, the designer may decide to group them together as sub-actions. With sub-actions just one button with a drop-down arrow is placed on the action bar. When users click the action they are presented with sub-actions. The mail template uses actions with sub-actions.



Procedure:
Creating sub-actions

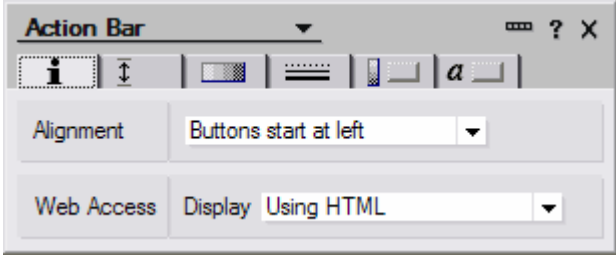
To create an action with sub-actions, complete the following steps:

Step	Action
1	From the Designer menu select Create -> Action -> Action with Sub Action .
2	In the Action pane, two entries are created. One for the action that will appear on the action bar and one for the first sub-action. They are both titled (Untitled) by default. 
3	Double-click the top-level action to give it a name. This is the name that will appear on the action bar. This action is not programmable; it serves only to group the sub-actions together.
4	Double-click the sub-action to give it a name and set its properties.
5	To create additional sub-actions, click once on the top-level action in the Action pane and select Create -> Action -> Action . This will create the new action as a sub-action.

Action Button and Action Bar Properties

Introduction In addition to the properties that you can set on the action buttons, there are properties that affect the appearance of the action bar itself. Action bar properties can control the appearance in the Notes client and from a web browser.

Procedure To set action bar properties, complete the following steps:

Step	Action
1	Open the action pane on the form where you want to set the properties.
2	From the menu choose Design -> Action Bar Properties .
3	<p>Select the desired options in the properties box.</p>  <p>There are several tabs in the Action Bar properties box that allow you to control the look and feel of the action bar, including:</p> <ul style="list-style-type: none">• Alignment• Bar height• Bar color• Borders• Button size• Button font size and style

Continued on next page

Action Button and Action Bar Properties, Continued

Action Bars in Web browsers

As mentioned earlier, action button can be very useful for providing Notes menu functionality to web browser users. Since browsers do not have the same menu commands as the Notes client, actions provide access to those commands.

The action bar can appear quite different from a browser client than in the Notes client. The action bar property “Web Access: Display Using Java applet” causes the action bar to more closely resemble its appearance in the Notes client from the web.

Example: Using HTML

Below is an action bar in a browser using the default setting “Web Access: Display using HTML”:



Example: Using Java applet

Below is the same action bar in Netscape using the setting “Web Access: Display Using Java applet”:



Exercise 10

Adding Actions to the Time Off Request Form

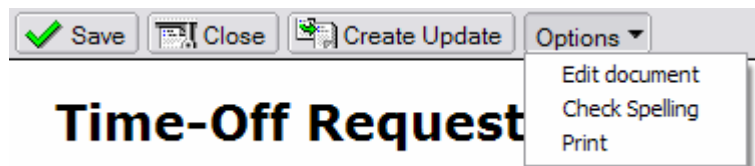
Introduction In this exercise you will add several actions to the Time-Off Request form.

Requirements & Resources This exercise should be completed in the Time Off Request database.

Instructions Perform the steps to complete the following:

1. Add action buttons to the Time-Off Request form to perform the following actions. Optionally, you may choose to create two or more of these actions as sub-actions:
 - Put the document into edit mode
 - Display the print dialog box
 - Create a new Request Update document
 - Check the document for spelling errors
 - Save the document
 - Close the document
2. Display all the actions on the action bar, using icons of your choosing.
3. Set the action and action bar properties for color and style of your choice. You may choose to experiment with several options.
4. Save the form.

Results After completing this exercise, you should have an action bar on the Time-Off Request for that resembles the following:



Lesson 11: Hide-When Options

Overview

Introduction Not all information on a form needs to be visible at all times. There may be circumstances under which you want to hide a certain field, action or piece of text. Notes provides hide-when properties for many elements within the database.

Objectives At the end of this lesson you will be able to:


- Apply built-in hide-when properties to text, fields and actions
- Apply a custom hide-when formula to text, fields and actions

In this lesson... This lesson contains the following topics:

- Hide-When properties
- Applying hide-when properties
- Hide-when properties for web applications

Hide-When Properties

Introduction

Hide-when properties are available for many elements within a database. If hide-when options are available the properties box for the elements will have the hide-when tab - . The hide-when options will work for Notes clients and web browsers.

Hide-When options for text and fields

Hide-When options on a form are applied on a paragraph basis. A new paragraph is created only when you press the Enter key. If text and a field are on the same line, that is in the same paragraph, you cannot hide one without the other. The text and field will have to be on different lines for them to be hidden independently of each other.

Hide-When options for tables

When hiding a cell of a table, you can hide one column of a row without the other columns of the same row being affected. For example, in a row with three columns, the first column can be hidden, the second and third will be displayed.

Hide-When for actions

Hide-When options can be very valuable for actions. Some actions do not work in certain situations, so having them available on the action bar can be confusing to the user. For example, an action that uses `@Command([ToolsSpellCheck])` requires that the document be in edit mode. A Hide-When option can be used to hide the action when the document is in read mode.

Built-in Hide-When options

There are several built-in Hide-When options to choose from:

- Previewed for reading
- Previewed for editing
- Printing
- Web browsers
- Mobile
- Opened for reading
- Opened for editing
- Copied to clipboard
- Notes 4.6 and later

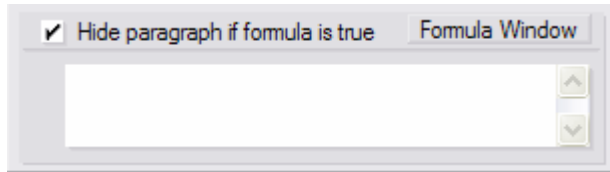
These options will vary depending upon the element that you are hiding. Actions, for example, will not have the options for Printed or Copied to the clipboard.

Continued on next page

Hide-When Properties, Continued

Hide-When formulas

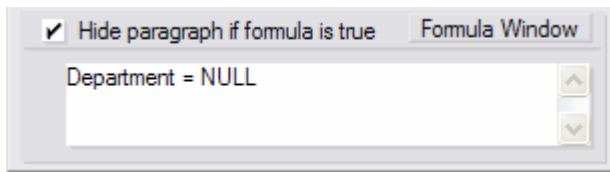
In addition to the built-in options you can write your own formula.



The Hide-When formula is similar to an If statement, but the true-action and the false-action are already built-in to the formula. You need only enter the criteria to be tested.


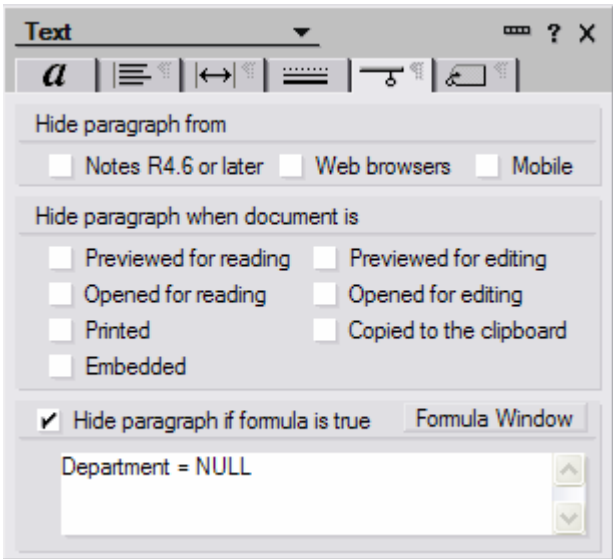
Example

To hide the Manager field if the Department field is blank, you would use the following formula:



Applying Hide-When Options

Procedure To set hide-when options for text and fields, complete the following steps:

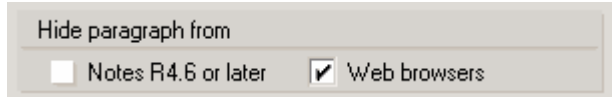
Step	Action
1	Access the properties box for the element that you wish to hide. If you are hiding multiple paragraphs, highlight the amount of text you want to hide. If you are hiding multiple actions, you will have to set the Hide-When options individually for each action.
2	Click the Hide-When tab - 
3	Select the desired Hide-When options. 

Text or Field? If you are hiding both text and fields in the same paragraph it does not matter whether you use the field properties or the text properties to hide the information. Both property boxes will show the same Hide-When options.

Hide-When Options on the Web

Uses of Hide-When for web clients

Hide-When options can be very useful when designing a single application that will be used on both Notes clients and web clients. There is a built-in option to hide information from web users.



Additionally, there are several @functions that can test the client environment such as @ClientType and @BrowserInfo. These functions can be used to determine which design elements to display to web clients.

Exercise 11

Applying Hide-When Options in the Time Off Request Form

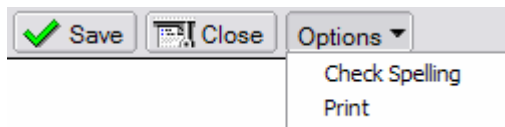
Introduction In this exercise you will apply Hide-When options to the Time Off Request form. You will hide some of the actions created in the previous exercise.

Requirements & Resources This exercise should be completed in the Time Off Request database. If needed, refer to the *Domino Designer 7 Help* database for assistance.

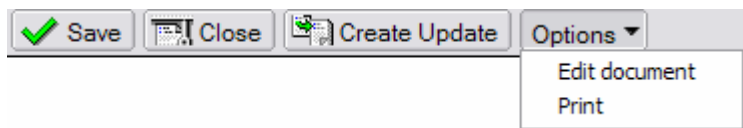
Instructions Perform the steps to complete the following:

1. Using the built-in Hide-When options, hide:
 - the Edit action button when the document is already in edit mode.
 - the Spell Check action button when users access the form from a web browser and when the document is in read mode in the Notes client.
 - the Print action button when users access the form from a web browser.
2. Using a formula, hide the Create Request Update when the Time Off Request has not yet been saved. *Hint* – test to see if the document is new.

Results The action bar should resemble the following in the Notes client in edit mode of a new request:



The action bar should resemble the following in the Notes client in read mode of an existing Time-Off Request document.



Lesson 12: Enhancing Form Layout with Tables

Overview

Introduction As you saw earlier, you can use basic tables to control the layout of text and graphics on a form. There are also specialized tables that can further enhance your form layout. In this lesson we will explore the following types of specialized tables:

- Tabbed
 - Nested
 - Animated
 - Programmed
-

Objectives At the end of this lesson you will be able to:

- Create and format a tabbed table
 - Nest a table inside another table
-

In this lesson... This lesson contains the following topics:

- Creating Tabbed Tables
 - Formatting Tabbed Tables
 - Creating Nested Tables
-

Creating Tabbed Tables

Introduction

A tabbed table, like any table, is a collection of columns and rows, but it displays just one row at a time, with the labels of the other rows displayed on tabs. The number of rows equates to the number of tabs. It allows the users to select which row to view by clicking on the tab. Tabbed tables are used extensively in the Domino Directory (NAMES.NSF). They are very useful for displaying large amounts of data in one form, without requiring the user to scroll down to see all the information.

Example

The following is an example of a tabbed table from the Domino Directory:

Administrators	Programmability Restrictions	Who can -
Full Access administrators:	Run unrestricted methods and operations:	
Administrators:	Sign agents to run on behalf of someone else:	
Database Administrators:	Sign agents to run on behalf of the invoker of the agent:	

Continued on next page

Creating Tabbed Tables, Continued

Procedure To create a tabbed table complete the following steps:

Step	Action
1	Position the cursor on the form where you want to create the table.
2	From the menu choose Create -> Table .
3	Enter the number of rows (tabs) required.
4	Enter the number of columns required.
5	For the table type, select the icon for tabbed table. <div data-bbox="630 720 1362 1415" data-label="Image"> </div>
6	Click OK.

Example:
New Tabbed
Table

The following is an example of a new tabbed table created with 4 rows (tabs) and 1 column:



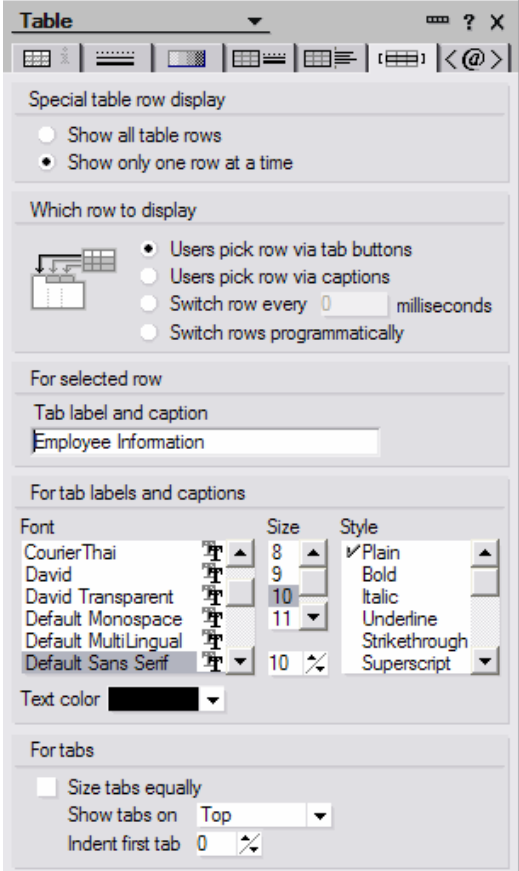
Formatting Tabbed Tables

Tabbed table properties

Once the tabbed table is created, you will need to set its properties to make it accessible to the user. A tabbed table uses most of the basic table properties, such as cell color and table border style and effects.

Procedure:

Complete the following steps to place labels on the row tabs:

Step	Action
1	Right-click on the table and choose Table Properties.
2	<p>Click on the 6th tab, Table Rows.</p> 
3	Click on the tab/row to be labeled and enter a label in “Tab label and caption text box.”
4	(Optional) Format the font and size of the tabs.
5	Repeat steps 3 through 6 for each tab/row.

Creating Nested Tables

Introduction

A nested table is simply one table placed inside another table. Basic and tabbed tables can be nested up to eight levels deep. Like basic tables, nested tables can be very useful for formatting the layout of text and fields on forms.

Procedure

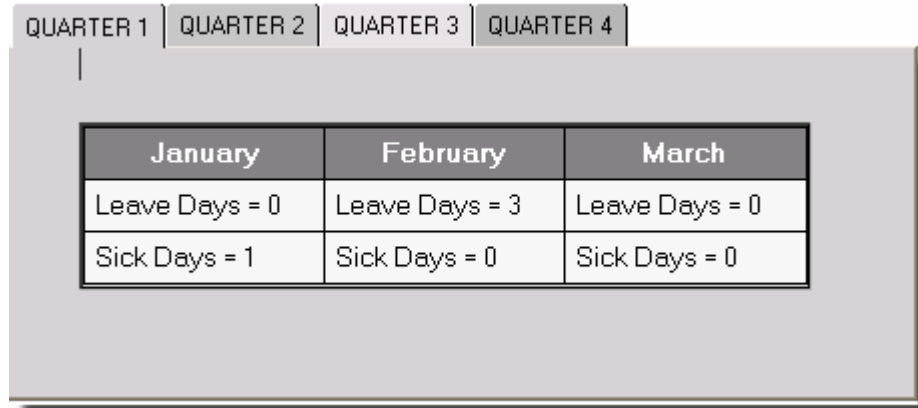
To create a nested table complete the following steps:

Step	Action
1	Create or access an existing table.
2	Position the cursor at the point where you want the new table.
3	Create the new table by choosing Create -> Table.
4	Select the type of new table, basic or tabbed.
5	Enter the number of rows and columns.
6	Click OK.

Note: Nested tables can also be created by pasting an existing table into another table.

Example: Nested tables

The following is an example of a basic table nested inside a tabbed table:



The image shows a screenshot of a software interface. At the top, there are four tabs labeled 'QUARTER 1', 'QUARTER 2', 'QUARTER 3', and 'QUARTER 4'. Below the tabs is a large rectangular area with a light gray background. Inside this area, a smaller table is nested. The nested table has three columns: 'January', 'February', and 'March'. The first row of the nested table contains 'Leave Days = 0', 'Leave Days = 3', and 'Leave Days = 0'. The second row contains 'Sick Days = 1', 'Sick Days = 0', and 'Sick Days = 0'.

Formatting Nested Tables

Each table has its own table properties such as cell color, border style and cell background. Nested tables should be formatted individually. When a table is pasted into another table both retain their formatting.

Lesson 13: Retrieving Data Stored in Saved Documents

Overview

Introduction When designing a Domino application it is always advantageous to reuse information whenever possible, whether that information is in the form of a design element or data that is stored in documents. In this lesson we will cover two @functions that allow you to access data stored in other documents.

Objectives At the end of this lesson you will be able to:

- Write an @DbColumn formula to access a column of data from a view in the current database or another database
- Write an @DbLookup formula

In this lesson... This lesson contains the following topics:

- @DbColumn formulas
- @DbLookup formulas

@DbColumn Formulas

Introduction The purpose of an @DbColumn formula is to look up and return an entire column of data from a view in the current database or another database.

A common use of @DbColumn formulas is to create a dynamic list as the options for a choice field such as radio buttons or a dialog list. If the list of options is generated by an @DbColumn formula then the designer does not need to maintain the choice field manually.

**@DbColumn
Syntax**

The syntax of the @DbColumn is:

@DbColumn(Class : "NoCache" ; Server : Database ; View ; ColumnNumber)

The following table describes the arguments of the function:

Argument	Description
Class : "NoCache"	Class indicates the type of database to be accessed. A Notes database can be indicated by "Notes" or "". NoCache specifies that the column data will be accessed each time the formula is executed. If NoCache is omitted the results are cached, stored for future use.
Server : Database	The server name and file name of the database
View	The complete, case sensitive name of the view that the formula will search. If the view has an alias, the alias should be used for this argument.
ColumnNumber	The column number that will be returned by the function. Notes starts counting columns from the left with 1. Not all columns are counted. For example, a column that holds constant data, such as a static text string, is not included in the count.

Continued on next page

@DbColumn Formulas, Continued

Specifying the Server and Database There are numerous ways to specify the server and database depending on the location of the database from which the column data is returned. The following table lists some alternatives for this argument:

If the	Then...
Lookup is performed in the same database as the @DbColumn function	the entire argument can be specified by "".
Database for the lookup is on a local server, but the database is not the current database	use "" for the server portion of the argument and the full database name "Filename.NSF"
Lookup is performed from the workstation to a database on a server	specify the name of the server and the database "ServerOne" : "Filename.NSF"
Lookup is performed from the work station to a database on a server and the formula needs will be executed from both the Notes client and web browsers.	using the database replica ID in place of both the server and database name allows you to access a replica copy of that database without having to specify either the server name or the database name. See the second example below. This method may be required in some circumstances when the formula will be executed from a web browser.

Examples: @DbColumn("", "" : "Lookups.NSF" ; "Departments" ; 2)

@DbColumn

This formula:

- uses a Notes type lookup
- caches the results
- looks in the Lookups database on the same server as the current database
- looks for a view named Departments
- returns column number 2

@DbColumn("", "NoCache"; "85255CEB:0032AC04"; "Requests"; 1)

This formula:

- uses a Notes type lookup
- does not cache the results
- looks for the database with the specified replica ID number
- looks for a view named Requests
- returns column number 1

Continued on next page


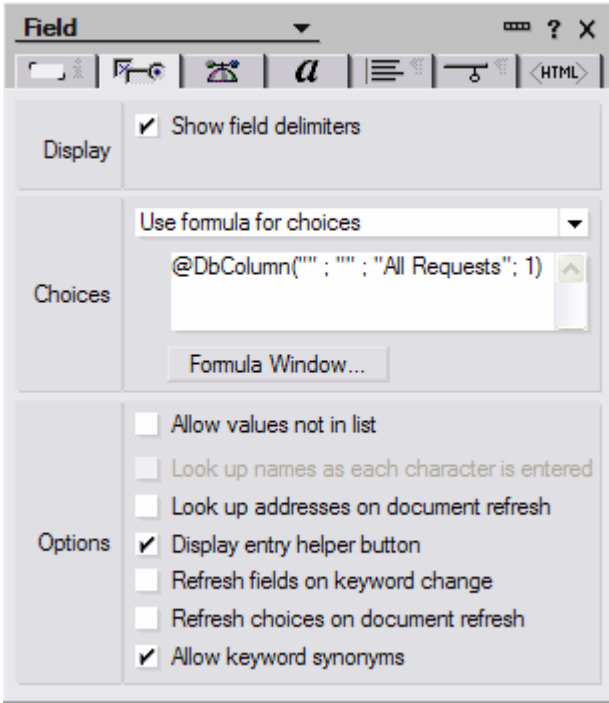
@DbColumn Formulas, Continued

@DbColumn formulas in choice fields

Using an @DbColumn formula in a choice field allows the user to pick from a dynamic list. When you design a choice field with an @DbColumn formula, you will not need to modify the design each time a choice needs to be added to or removed from the list. Because the view that the @DbColumn uses will immediately reflect newly added or deleted documents, the user will be selecting from the latest choices.

Procedure

To enter an @DbColumn formula in a choice field, complete the following steps:

Step	Action
1	Create or access a choice field, such as radio buttons or dialog list.
2	In the field properties box, select the Control Tab - 
3	For choices select the option "Use formula for choices"
4	Enter the formula in the space provided.  If you would like a bigger area for typing the formula, click the button for Formula Window.

Exercise 13.1 - Optional

Using @DbColumn Formulas in the Time Off Request Form

Introduction In this exercise you will modify the existing DepartmentName field to use an @DbColumn formula to retrieve a list of departments from another database.

Requirements & Resources The database Department Information (DeptInfo.NSF) will be used for the database argument in the @DbColumn formula.

Instructions Complete the following steps:

1. Make sure you have a local copy of the Department Information database in your local Notes data directory.
2. Open the Department Information database from the Notes client.
3. Modify at least one of the existing documents by changing the current manager's name to your own user name as it appears in your ID file. For example, John Smith/DLand. (This is being done to facilitate a future exercise). Save the changes.
4. On the Time-Off Request form in the Time-Off Requests database, modify the existing DepartmentName field to "Use a formula for choices".
5. Enter an @DbColumn formula that uses the Department Information database and the Departments view. The Departments view uses the alias Dept.
6. Make sure the DepartmentName field property "Refresh fields on keyword change" is selected.
7. Save the changes.
8. Test the @DbColumn formula in the Notes client.

Result At the end of this exercise when you create a new Time Off Request document and click on the down arrow for the Department Name, you should see the names from the Departments view of the Department Information database.

@DbLookup Formulas

Introduction

Given a key value, @DbLookup looks in the first sorted column of the specified view and finds the document containing the key value. It then returns information from the document that is either in a specified column of that view or the contents of a specified field.

@DbLookup Syntax

The syntax for the @DbLookup function is very similar to the @DbColumn function. In fact, @DbLookup has just one additional argument.

@DbLookup(Class : "NoCache" ; Server : Database ; View ; Key ; ColumnNum)

OR

@DbLookup(Class : "NoCache" ; Server : Database ; View ; Key ; "Fieldname")

The following table describes the arguments of the function:

Argument	Description
Class : "NoCache"	Class indicates the type of database to be accessed. A Notes database can be indicated by "Notes" or "". NoCache specifies that the column data will be accessed each time the formula is executed. If NoCache is omitted the results are cached, stored for future use.
Server : Database	The server location and file name of the database
View	The complete, case sensitive name of the view that the formula will search. If the view has an alias, the alias should be used for this argument.
Key	The piece of information that is looked up in the first sorted column of the view. The key identifies the document to be read. The key is usually a field name.
ColumnNumber	When you use a column number, Notes finds the document in the view that matches the specified key, and returns whatever value is <i>displayed</i> in the indicated column for the document.
Fieldname	The name of the field from which the data will be retrieved, once the correct document(s) has been identified. If you use a field name the field does not need to be included in the view. Unlike most formulas that use field names, the fieldname argument is enclosed in quotation marks "";

Continued on next page

@DbLookup Formulas, Continued

Example::
Using
@DbColumn
and
@DbLookup

A database has a form named Department Information. This form stores department information in the following fields:


Field name	Stores..
Department	department names
DepartmentManager	manager of the department
CostCenter	the cost center of the department

When a document created with the Department Information form is saved, it is displayed in a view called Departments with an alias of Dept. The Dept view has the following:

Column Number	Displays
1	Department
2	DepartmentManager
3	CostCenter

The database also has a form named Employee Information with the following fields:

Field name	Description
EmployeeDept	A Dialog List field that uses @DbColumn to return column 1 from the All Departments view. @DbColumn(“” ; “” ; “Dept” ; 1)
EmployeeManager	A computed field that uses an @DbLookup to retrieve the department manager using the contents of the EmployeeDepartment field as the key. @DbLookup(“” ; “” ; “Dept” ; EmployeeDept; 2)

After clicking the entry button  for the DepartmentManager field, the user will be presented with a list of departments from the Departments view. Once the user makes a selection from the list, the EmployeeManager field will be automatically populated with the appropriate department manager as a result of the @DbLookup function.

Exercise 13.2 - Optional

Using @DbLookup functions on the Time Off Request Form

Introduction In this exercise you will modify the ManagerName field in the Time Off Request form to use an @DbLookup formula. This formula should use the value selected in the Department field to lookup the name for the manager.

Requirements & Resources As in exercise 13.1, you will use the Department Information (DeptInfo.NSF) database and the Departments view for the @DbLookup formula.

Instructions Complete the following steps:

1. On the Time-Off Request form modify the ManagerName field to be Computed.
2. In the ManagerName field use an @DbLookup formula to return the manager's name from the Department Information database using the department stored in the Department field as the key value. The name of the manager is not displayed in the Dept view, so you will need to include the field name in the formula.
3. Test the form.
4. Does it work? What error message do you receive? What needs to be added to the formula for ManagerName to make it work correctly?

Hint: When the form is first created the DepartmentName field is blank. Modify the @DbLookup formula to test for an empty DepartmentName field.

Checking your results After completing this exercise when you select a department name in the Time-Off Request form, the name of the manager should be populated for you.
A completed example is available in the Time-Off Requests Part 2 database.

Lesson 14: Implementing Work Flow

Overview

Introduction One of the most powerful features that a designer can add to a database is workflow. Workflow applications are those that route documents from one database to another and/or send notifications to users' mail files. In this lesson workflow is introduced into the Time Off Request database. Workflow applications can become quite complex. In this lesson we cover only the basics of application workflow and routing to user's mail files. For information on mail-in databases refer to the *Domino Designer 7 Help* database.

Objectives At the end of this lesson you will be able to:

- Enable a form for mail routing.
- Create an action that uses an @MailSend function with arguments to send notifications to mail files.

In this lesson... This lesson contains the following topics:

- Workflow applications
- Mail enabling a form
- The @MailSend Function

Workflow Applications

Introduction Workflow can be defined as the activities and interactions required to carry out a business task or process. In the Time Off Request application we want:

- the manager to receive a mail notification of a pending request when the user saves the document in the database.
- the user to be notified when the manager has approved or denied a request.

Added value A workflow application can add value to the process with the following functionality:

- Tracking the documents in the stages of the workflow
- Managing the flow of documents
- Providing reporting tools

Who does what In a workflow application it is very important to keep in mind which users are going to be performing which activities. For example, in the Time Off Requests database, we will only want department managers to approve pending time off requests. Features to control who can approve the requests will be covered later.

When is the document sent Document routing can be triggered by:

- User actions, such as clicking a button
- Field formulas
- Form events
- Workstation or server agents

Requirements for routing For a document to be successfully routed it must:

- Have an addressee(s) specified
- Be created with a mail enabled form

Mail Enabling a Form

Introduction

Before a document can be routed automatically the form that it was created with must be mail enabled. While any document can be forwarded using the menu command **Actions -> Forward**, this is not considered automatic mail routing. The three ways to mail enable a form covered in this lesson are:

- The form property “On close: Present mail send dialog”
 - @MailSend without arguments
 - @MailSend with arguments
-

SendTo field

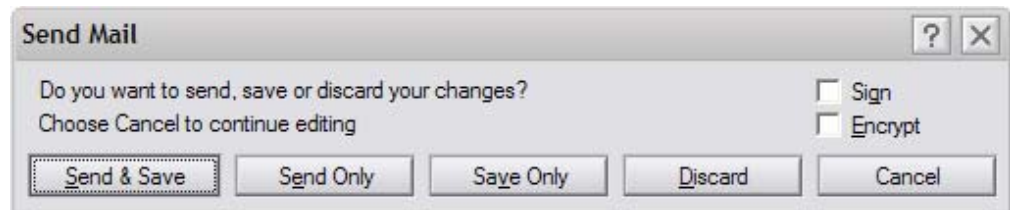
SendTo is a reserved field name that is commonly used on forms that are meant to be routed. The purpose of the SendTo field is to store the name of the recipient(s) of the routed document.

Because SendTo is a reserved field name, you should not use the name SendTo except for fields that will serve this specific purpose. The SendTo field must contain the name(s) of a person, group or mail-in database listed in the Domino Directory.

For details other reserved field names, refer to the *Domino Designer 7 Help* database.

Mail Send Dialog box

You are probably familiar with the mail send dialog box from your own mail file. When the form property “On close: Present mail send dialog” is selected on the Defaults tab of the form properties box the following dialog box is presented when a user closes a document:



Requirements:

This method of mail enablement requires that the form contains a SendTo field. If it does not contain a SendTo field, you will be presented with the dialog box, but choosing an option that includes sending will result in an error.

Results:

If the user selects either of the first two choices in the dialog box the entire document is routed to the recipient(s) contained in the SendTo field.

The @MailSend Function

@MailSend without arguments

@MailSend without arguments is commonly placed in action buttons or accessed via a hotspot. The function mails the current document (the one being processed when the @function is evaluated) to the recipient(s) designated in the document's SendTo field. The document must have a SendTo field with a valid recipient name.

@MailSend is not supported in web applications.

“Form not found”

By default, when a document is saved the data is stored separately from the form. The form name is referenced in the field Form, which is automatically created by Notes. Under these circumstances when a document is mailed, the form is not mailed with it. When the recipient attempts to open it, if the database that he is working in (frequently his own mail file) does not have the same form that the document was created with, he will receive the error message “Form Not Found: *Form Name*”. The document will then be opened with the default form of the database, if one has been designated.

The form property “Store form in document” is one way to prevent any problems in opening the document. Using this property ensures that when the recipient opens the document in the destination database, such as a mail file, it will be displayed with the proper form.

Continued on next page

The @MailSend Function, Continued

@MailSend with arguments

@MailSend with arguments works quite differently than without arguments. Rather than send the current document @MailSend with arguments composes a new mail memo using the arguments supplied in the function.

@MailSend(SendTo ; CopyTo ; BlindCopyTo ; Subject ; Remarks ; BodyFields ; [Flags])

The following table describes the @MailSend arguments:

Argument	Description
SendTo	The primary recipient(s) of the mail memo. This SendTo argument does not need to be a SendTo field. For example, if the form has a field named Manager, the Manager field can be used for the SendTo argument. This is the only required argument.
CopyTo	(Optional) The copy recipient(s) of the mail memo.
BlindCopyTo	(Optional) The blind copy recipient(s) of the mail memo.
Subject	(Optional) The text you want displayed in the Subject field. This is equivalent to the Subject field on a mail memo; the message is displayed in the Subject column in the views in the recipients' mail databases.
Remark	(Optional) Any text you want at the beginning of the memo's body field.
BodyFields	(Optional). The names of one or more fields from the current document that you want included in the mail memo. The fields are appended to the memo in the order in which you list them.
Flags	(Optional) One or more flags indicating the memo's priority and security.

Continued on next page

The @MailSend Function, Continued

Flags

The following table lists and describes the flags for the @MailSend function:

Flag	Function
[IncludeDocLink]	Includes a link pointing to the document that was open or selected when @MailSend was used. You must include this flag if you want that document linked to the mail memo.
[Sign]	Electronically signs the memo when mailing it, using information from the user's ID. Signing will not occur unless you include this flag.
[Encrypt]	Encrypts the document using the recipient's public key, so that only the recipient whose private key matches can read the document. Encryption will not occur unless you include this flag.
[PriorityHigh]	Immediately routes the message to the next-hop server as defined by the combination of Mail Connection records and server records.
[ReturnReceipt]	Notifies the sender as each recipient reads the message.

Example

The following @MailSend function uses temporary variables:

```
send := Manager;
cc := NULL;
bcc := NULL;
subject := "Pending Request";
remark := "Follow this link to a Time-Off Request that requires approval - ";
bodyfields := NULL;
flag := [IncludeDocLink];

@MailSend(send ; cc ; bcc ; subject ; remark ; bodyfields ; flag)
```

Note: Remember that the temporary variables only exist for the duration of the formula evaluation. The values are not saved anywhere.

Exercise 14 - Optional

Implementing Workflow in the Time Off Request Database

Background Currently when a user creates and saves a Time Off Request it is stored in the database. The managers must look in the database to determine if there are any requests awaiting their approval. In this exercise you will implement mail routing on the Time-Off Request form so that when a user creates a Time-Off Request, she should be able to click an action button to submit the form to the manager for approval. Additionally, when the manager approves or rejects the request, the manager should be able to click an action button to send a notification to the user regarding the status of the request.

For this exercise, you will play the role of both requestor and manager so that you can verify your own work.

Requirements & Resources This exercise will require two new action buttons on the Time Off Request form. It will require you to check your mail file for the notifications. If you do not have a server-based mail file, you will not be able to complete this exercise.

- Instructions** Complete the following:
1. Add a Submit for Approval action button to the form. The action should do the following
 - Send a mail memo to the manager
 - Specify “Pending Time-Off Request” as the subject
 - Include remarks of your choice in the formula.
 - Include a link to the actual Time-Off Request document
 2. Add a Notify Requestor action button to the form. The action should do the following:
 - Send a mail memo to the requestor
 - Specify “Updated Time Off Request” as the subject
 - Include remarks of your choice in the formula.
 - Include a link to the actual Time Off Request document
 3. Test the form in the Notes client using a document for which your name appears in the ManagerName field. You may need to create one.
-

Continued on next page

Exercise 14, Continued

Implementing Workflow in the Time Off Request Database

Results

When you click on a Submit for Approval button in the Time Off Request form, a mail memo should be routed to the person listed in the Manager field. When you, acting as the manager open the mail memo, you should be able to click on a document link that takes you to the actual request document in the Time Off Requests database.

Continuing to act as the manager, when you change the status of the request and click on the Notify Requestor button, a mail memo should be routed the person in the requestor field. When you, now acting as the requestor, open the mail memo, you should be able to click on a link that takes you to the actual request document.
