EntwicklerCamp 2012

Turbocharge Development in Notes/Domino 8.5 – with Formulas!

By *Rocky Oliver*
Collaborative Technologies Geek

rocky.oliver@gmail.com
www.linkedin.com/in/lotusgeek

Introduction

- Began in Lotus-land in 1992
- Worked for large consultancies, pre-ipo startups, and out on my own
- Worked for Lotus/IBM – twice!
- Written a couple of books
  - Latest was “Notes and Domino 6 Programming Bible” (Wiley, 2003)
- I'm available – HIRE ME!

Formulas - Background

- Formulas are the oldest language in Notesdom
- Based on the same API as all other scripting languages
- Runs much, much faster - “closer to the iron”
- Provides shortcuts to powerful functionality
- Is outstanding at processing lists of stuff
- Continues to be updated in N/D

Formulas – Where to Use?

- Actions, buttons, hotspots, etc.
- Computed values
- Hide/when formula
- Window titles
- Default field values
- Column formula
- Section editor
- Input validation
- Input translation
- View selection formula
- XPages!
- REMEMBER: Not all formulas can be used in all situations!

Formula – Input Validation

- Input Validation formulas are used to validate the value(s) entered into a field
- Provide conditions to define success and failure, in a variety of ways
- Can also be used to cause other values to change based on validation condition
- Primary @functions used are @If, @Failure and @Success

Formula – Input Validation

- Use @Success to indicated value is correct
- Use @Failure to reject the value and provide a message indicating so to the user
- @Success signals a True condition, which indicates validation was successful; @Failure signals a False condition, indicating the validation was not successful
- Validation formula is triggered when form is refreshed or a save is attempted
- @Failure will cause the refresh or save to fail, and return the cursor to the failed field (if not hidden)

Formula – Input Validation

- You can use @ThisField and @ThisValue to reuse the same Validation Formula code across multiple fields

```
@If(@ThisValue = “”; @Failure(“You must provide an answer for the field “ + @ThisField + “.”); @Success)
```

Formula – Input Validation

- You can prompt the user once for all fields that fail validation, if desired
  - Create a hidden, Computed For Display field at the bottom of the form
  - Use a formula like this (simple version – better one later):

```
@If(@DocIsBeingSaved & (Name = "" | Occupation = "");
@Failure("Please make sure all fields have a value");
@Success)
```

Formula – Input Validation

- The highlighted @function, @IsDocBeingSaved, allows you to have formulas (such as Input Validation) that only fire when the document is actually being saved
- Not using @IsDocBeingSaved will irritate your customers, thereby *causing you **pain and misery***

Formula – Input Validation

- Validation formulas don't have to be simplistic; you can use them to do a variety of validation types.
  - Checking for a value or range of values:

```
@If(@ThisValue < 5 | @ThisValue >10; @Failure("You must
  enter a value between 5 and 10"); @Success)
```

  - Even fairly complex validations, like U.S. Phone Numbers:

```
@If(@Matches(@ThisValue; "{0-9} {0-9} {0-9} {-} {0-9}
  {0-9} {0-9} {-} {0-9} {0-9} {0-9} {0-9}"); @Success;
  @Failure("You must enter a U.S phone number in the
  format of xxx-xxx-xxxx.")
```

Formula – Input Translation

- Field Translation Formula is used to modify the input value of a field into a format that is usable for either a person or code
- Could be a simple as `@Trim(@ThisValue)` to remove extra spaces
- Can also be used to do “quick fixes” on user-provided data so a failed validation isn't necessary

Formula – Keep These in Mind

- @ReplaceSubstring is another excellent function at string manipulation, as it provides find and replace functionality on a string
  - Remove symbols from a string in a field

```
symbols := "!" : "@" : "#" : "$" : "%" : "^" : "&" :
"*" : "+" : "=";
result := @Trim (@ReplaceSubstring (somefield1;
symbols; ""))
```

  - Replace backslash with frontslash, such as URLs

```
@ReplaceSubstring (@ThisValue; "\" ; "/")
```

  - Replace new lines with spaces

```
@ReplaceSubstring (@ThisValue; @NewLine; " ")
```

Formula – Keep These in Mind

- @LowerCase, @UpperCase – great to make all values consistent for comparison
  - e.g. FOObar, fooBAR, and FoObAr are all the same using @LowerCase (“foobar”)
- @IfError(... == @If(@IsError(...
- @Word – GREAT when used to obtain values from a concatenated string or list of strings (think: “column”)
  - Example

Formula – Keep These in Mind

- @Random – generates a random number from 0 to 1, inclusive
  - Use formula from Help to generate a between any two numbers:

```
( y - x ) * @Random + x
```

- Example time!

Formula – Keep These in Mind

- Working with numerical values can be particularly difficult – even in formulas
  - 3.33 * 3.33 == 10.89 is easy
  - 3.33333*3.33333 == 11.1110888889... not so much
- @Round allows you to make decimal places consistent
  - Caveat: @Round does NOT work the same as LS Round!
  - @Round(7.7255; 0.01) == 7.73 (nearest decimal place)
  - Round(7.7255, 2) = 7.73 (# of decimal places)

Formula – These are Handy

- @ThisValue and @ThisName
  - Used in field Translation and Validation formulas
  - @ThisName == the name of the current field
  - @ThisValue == the value of the current field
  - (I've been using these earlier in the session)
- Allow for easy reuse of code between fields (i.e. you can copy/paste them from one field to another
- These are not usable outside of the Validation and Translation formula context

Formula – These are Handy

- @SetEnvironment and @Environment
  - Allows you to set your own environment variables OR system variables in the user's NOTES.INI
  - Quick-n-dirty, old school way to pass information back and forth between LotusScript and Formulas
  - @Environment can be used to both set AND retrieve an environment variable
    - Best Practice? @SetEnvironment for setting, @Environment for getting; it makes the code easier to understand

Formula – Date/Time Manipulation

- @Adjust allows you to change a date/time value, from year to second
  - Combination of all “Adjust...” functions in LotusScript
  - @Adjust(dateVar; Y; M; D; H; M; S)
  - If changing more than one param, @Adjust evaluates RIGHT to LEFT (i.e. from Second, through to Year)

Formula – Working with Text Lists

- One of the Formula Language's true strengths is text manipulation
- There are many, many formulas that work with both single values and text lists
  - @Word was an example earlier
- Here are a few others that are helpful...

Fun with Lists – Trim and Replace

- @Trim – does what it says, i.e. removes leading, trailing, and extra spaces from a string or list of strings
- @Replace... replaces items in a source list that match items in a second list with items in a third list to produce a new list (more on this later)
- @Trim(@Replace – very powerful for list manipulation
  - Example time!

Fun with Lists - @Unique

- @Unique leads two lives
  - With no parameters it generates a unique alphanumeric string
  - When provided a list of values it removes duplicates and returns a list of unique values

Fun with Lists - @Sort

- @Sort is not only capable of standard sorting, it also has unique abilities for “unconventional” sorting as well
- You can sort ascending, descending, case sensitive/insensitive, and so on – the normal stuff
- You can also provide a *custom sort formula*
  - Use $A to represent the current value, and $B to represent the next value in the list
  - The formula is evaluated – if True, it does not change the order; if False it swaps them
  - Then $B becomes $A, and the next value becomes $B
- Example Time! (This should make it easier to understand...)

@Transform and @For

- @Formulas have the ability to “loop” now, like While, For, and Forall statements in LS
- @For – works identically to the JavaScript version of For
- @Transform – think of it as “@ForAll”, as this will help you keep it straight

@For – the Basics

- Syntax for @For:

  @For(start; evaluation; iteration; formula)

  - Start == the starting number for the evaluation
  - Evaluation = the condition to evaulate whether another loop is warranted; if true, run the formula again; if false, end the loop
  - Iteration == the increment by which the start number should be incremented
  - Formula == the number that is executed on every loop

@For - Example

- @For(n := 1; n <= @ Elements (customers); n := n + 1; custmail[n] := @ReplaceSubstring(customers[n]; " "; ".") + "@customerdomain.com");

  custmail

- This simply iterates through a list of customers and creates an email for each of them by replacing any spaces with periods, and appending "@customerdomain.com" to them; the resulting list is assigned to the custmail variable

@Transform - Syntax

- Syntax:
  @Transform(someList; varName; formula)
- SomeList == the list of values to be modified
- VarName == a STRING of the variable that represents the current list member being processed
- Formula == the formula being applied to the list member
- Examples

@Transform – the Basics

- @Transform is basically @Forall
- It takes a list and allows a formula to be run against every member of a list
- It is very, very fast

View Selection Formulas

- Couple of pointers on View Selection Formulas
  - You cannot use @DbColumn or @DbLookup in View Selection Formulas (although it would be nice)
  - Do not use Time values (like @Today) – HUGE performance issues
  - Use @AllDescendants instead of @IsResponseDoc

Formula – Let's Talk about @Text

- @Text, on the surface, simply ensures that a value is a text value – and if not, it converts the value to text; but it does more
- A 2nd parameter can be provided to @Text to format a date/time value and present it a number of ways
- A 2nd parameter can be provided to @Text to format a numeric value many ways as well
- Help has all of the various codes for the 2nd parameter

@Text – One More Thing

- Often have problems getting numeric values from fields, even when field is “number”
- “Type cast” the number to make sure it's a number by using @TextToNumber(@Text (numField))
- Same thing for date fields - @TextToTime (@Text(datetimeField))

@Matches

- @Matches determines if a value matches another value, or a provided pattern
- True is returned if the first value matches the second, or the pattern provided in the second parameter
- Allowed wildcards and patterns are in Designer Help
- Example: Earlier demonstrated checking a U.S. phone number (xxx-xxx-xxxx)

```
@Matches(@ThisValue; "{0-9} {0-9} {0-9} {-} {0-9} {0-9} {0-9} {-} {0-9} {0-9} {0-9} {0-9}")
```

Formulas in XPages

- XPages are now available in Domino 8.5.x
- @Formula is available to use in Server Side JavaScript (SSJS)
- Not all @Formula are supported in Xpages
- Three syntactic changes to use @Formula in XPages:
  - Use commas rather than semicolons
  - Use exact case
    - Example: var uname = @Name("[CN]", @UserName())
  - NULL should be used in place of 0 for formulas (such as @Adjust discussed)

Formulas in XPages

- Unfortunately the Designer Help is not very useful
  - Very few examples for Xpages formulas
  - Syntax stuff omitted from descriptions
  - EXAMPLE: Quotes are needed on formula keywords - @Name(“[CN]”, name) – needs those quotes
  - Parens are needed on all formulas, even if it takes no parameters - @UserName() – needs those parentheses