

THE VIEW

LOTUS DEVELOPER 2006

The Top 30 LotusScript Development Tips

Bill Buchan
HADSL

© 2006 Wellesley Information Services. All rights reserved.



What We'll Cover ...

- **Introduction**
- Nursery Slopes
- Everyday Tips
- Advanced Tips
- Wrap-up

Introduction

- What is this about?
 - ◆ This talk aims to run through 30 different LotusScript tips
 - ▶ 10 of these tips (Nursery Slopes) are fundamental
 - ▶ 10 of these tips (Everyday Tips) are less well known
 - ▶ 10 of these tips (Advanced Tips) are there to provoke thought

Introduction (cont.)

- **Why 30 tips?**
 - ♦ “The Lazy Programmer” methodology
 - ♦ Experience and understanding
 - ♦ Push your comfort zone and provoke thought
- **Who am I?**
 - ♦ Bill Buchan
 - ♦ Dual PCLP in v3, v4, v5, v6, v7
 - ♦ 10+ years senior development consultancy for Enterprise customers
 - ▶ **Learn from my pain!**
 - ♦ 5+ years code auditing
 - ♦ CEO – HADSL – developing best-practice tools

What We'll Cover ...

- Introduction
- **Nursery Slopes**
- Everyday Tips
- Advanced Tips
- Wrap-up

NS #1: Option Declare

- You should always use “Option Declare”
- Yes, but why?
 - ◆ If not, all variables are created at runtime as variants
 - ▶ This makes type-checking redundant
 - ▶ Data conversion costs 10x performance!
 - ▶ And means all errors will be runtime errors
 - ◆ Remember: Test lots, test early
 - ◆ Use the strengths of the compiler to help you

NS #2: Templates and Versions

- In terms of custom code, you should always
 - ♦ Create templates with new code
 - ▶ Databases with “ntf” extension
 - ▶ Or databases with Master Template Name set
 - ♦ Create a separate copy for each version
 - ♦ Keep them in a central repository
- Why?
 - ♦ It makes it far easier to roll back
 - ♦ Version control
 - ♦ Easier to construct test cases with “other” versions

NS #3: Application Lifecycle

- You should always:
 - ◆ Develop in a development “sandbox”
 - ◆ Test in a user acceptance environment
 - ◆ Pass the template to your administrator to apply
- Why?
 - ◆ It’s a basic change control system
 - ◆ Professionalism
 - ◆ It shakes out “hardcoded” bugs
 - ▶ Hardcoded server names, paths, replica IDs
 - ◆ It allows you to run extensive testing without affecting production

NS #4: How to Code

- How to code?
- Code for maintenance
- Only code for performance if required
 - ♦ Get it working before you get it working for speed
- Why?
 - ♦ The highest cost in development is software maintenance
 - ♦ Make it easier for the person maintaining the application
 - ♦ It could be YOU



Note

NS #5: "Measure Twice, Cut Once"

- Spend more time thinking and less time coding
- Try to think of at least TWO methods of solving a problem
 - ♦ The best approach is usually a blend of the two
 - ♦ Think about the data model!
- Why?
 - ♦ More planning, less work

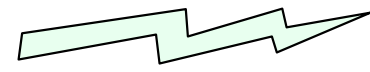
"You know it's been a rough night when you wake up next to some code you don't recognize"
- Bob Balaban



NS #6: Extending Arrays the Easy Way

```
Sub initialize()  
    Dim myArray() as String  
    redim myArray(0)  
    call myExtend (myArray, "Hello Sailor")  
end sub  
  
function myExtend(S() as String, ns as String) as integer  
    if (S(ubound(S)) <> "") then redim preserve S(ubound(S)+1)  
    S(ubound(S)) = ns  
    extend = true  
end function
```

- Pros:
 - ♦ No need to keep separate index of array size
 - ♦ Automatically "trims"
- Cons:
 - ♦ Slow with large arrays
 - ♦ Need to define some "empty" value



GOTCHA!

NS #7: The List Operator

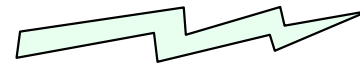
```
Dim WR list as String
WR("Peter") = "Perfect"
WR("Penelope") = "Pitstop"
```

```
Print "Peter's last name is: " + WR("Peter")
if not isElement(WR("Dick")) then print "Dick isn't racing!"
forall thisRacer in WR
    Print listtag(thisracer) + " " + thisRacer
end forall
```

- List stores a value with a unique lookup key
 - ♦ Pros:
 - ▶ It's easy and it's fast
 - ▶ It's built right into LotusScript, since v4.5
 - ♦ Cons:
 - ▶ You can't directly read and write a list from a document item
 - *Convert to an array first*

NS #8: Logging

- If you have applications with scheduled Agents
- Or if you have a diverse range of clients
 - ♦ Then you need to log your Agent (both client and scheduled) runtime status
- Why?
 - ♦ Applications will break
 - ♦ You need to know when they break
 - ♦ Timing metrics for performance testing
- Beware!
 - ♦ Don't make the logging so slow that it affects application performance!



GOTCHA!

NS #9: Code Structure

- **Problem Decomposition:**
 - ◆ Smaller code sequences are easier to:
 - ▶ **Maintain – the majority of the cost**
 - ▶ **Reuse – if kept simple**
 - ◆ Rule of thumb: If it's more than a screenful – can it be broken up?
 - ▶ **But – don't be prescriptive**
- **It's all about**
 - ◆ Problem decomposition
 - ◆ Maintainability
 - ◆ Code re-use
 - ◆ Get the data model right to start with!

NS #9: Code Structure (cont.)

- **Tips:**
 - ◆ Keep functions/procedures/classes at a manageable size
 - ▶ **But don't be prescriptive**
 - ◆ Think before decomposing problems
- **Discussion**
 - ◆ "Declare at the top" or "Declare in the code"
 - ▶ **I prefer to declare variables in the code**
- **Comments**
 - ◆ Don't over comment – explain the "Why"!
 - ◆ I usually only comment non-intuitive algorithms
 - ▶ **"This isn't what you think ... "**
 - ▶ **Comment pseudocode to start with**

NS #10: Code Hiding

- **Code hiding**

- ♦ Means decomposing your code so that consumers don't see your code
 - ▶ **Including yourself**
- ♦ Infers a logical interface (with clear naming!)
- ♦ Customers/peers now write to the "interface"
 - ▶ **Not the code itself**
- ♦ It simplifies the coding experience
- ♦ How?
 - ▶ **"Private/Public" methods in classes, script libraries, etc.**

What We'll Cover ...

- Introduction
- Nursery Slopes
- **Everyday Tips**
- Advanced Tips
- Wrap-up

ED #1: Error Trapping

- If I hear “I write perfect code. I don’t need error trapping,” I think of two things
 - ♦ “Yeah, right”
 - ♦ “Fire this guy”
- Error handling is mandatory. Seriously.
- Now, how? Two routes:
 - ♦ Single error handler at the top, bubble errors up
 - ▶ Simple, but difficult keeping context
 - ♦ Error handler implemented at function level
 - ▶ More work, but much more granular

ED #2: Defensive Coding

- Defensive coding is:
 - ♦ Assume the worst, check all input
 - ▶ On every function
 - ♦ Does this affect performance? Not usually
 - ♦ A typical function looks like:



```
Function mytest(p1 as String, p2 as String) as
integer
    mytest = false
    if p1="" then exit function
    if p2="" then exit function
    ...
    ' Now actually do something!
    ....
    mytest = true
end function
```

ED #3: Protecting Code

- Commercial applications need to hide their code. How?
 - ♦ Create a template and “hide design”
 - ▶ Pro: Easy
 - ▶ Con: You may allow form customization, etc.
 - ♦ Or remove LotusScript source code from script libraries
 - ▶ Use NotesNoteCollection to find script design document
 - ▶ Replace item “\$ScriptLib” with a String – “Hello”
 - *Con: Not so simple*
 - *Pro: Other design elements can be modified*
 - ▶ Don’t do this on your development copy ...



Heads Up!

ED #4: Use NotesDateTime Instead of Strings!

- Don't store date/time values as Strings
- Use NotesDateTime structures, and save them
- Why?
 - ♦ You don't know how the client will interpret dates ...
 - ▶ Is it dd/mm/yyyy or mm/dd/yyyy?
 - ♦ It means that views will be able to sort on dates
- This happens more often than you think!

ED #5: DXL as Transport

- Consider the situation where your customer wants to send back “log” documents easily
 - ◆ Use a LotusScript Agent to:
 - ▶ Pick up all selected documents
 - ▶ Create a Memo with a Rich Text Field
 - ▶ Use DXL to store the documents in the RT field
 - ◆ At the receiving end:
 - ▶ Unpack the mail message to a DXL Stream
 - ▶ Construct new documents to store the data
 - ◆ This is data transfer without replication



ED #5: DXL as Transport Example Code – Send

```
Dim sSession As New NotesSession
Dim dbThis As notesDatabase
Set dbThis = sSession.CurrentDatabase
Dim dc As NotesDocumentCollection
Set dc = dbThis.UnprocessedDocuments
```

```
If (dc Is Nothing) Then exit sub
If (dc.count < 1) Then exit sub
```

```
Dim doc As NotesDocument
Set doc = dc.GetFirstDocument
```

```
While (Not doc Is Nothing)
```

```
    Dim de As NotesDXLExporter
    Set de = sSession.CreateDXLExporter( )
    Call de.setInput(doc)
```

```
    Dim dxl As String
    dxl = de.Export
    ' continued overleaf..
```



Actual Work

ED #5: DXL as Transport Example Code – Send (cont.)

```
Dim dbMail As New NotesDatabase("", "")
Call dbMail.OpenMail()

Dim docM As NotesDocument
Set docM = dbMail.CreateDocument

Call docM.ReplaceItemValue("Form", "Memo")
Call docM.ReplaceItemValue("Recipients", "logs@hads1.com")
Call docM.ReplaceItemValue("SendTo", "logs@hads1.com")

Call docM.ReplaceItemValue("Subject", "Log Documents")

Dim rt As New NotesRichTextItem(docM, "Body")
Call rt.AppendText(dxl)

Call docM.Send(False)

Set docM = Nothing
Set de = Nothing

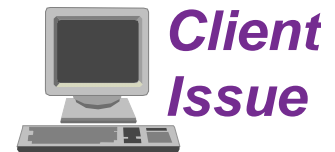
Set doc = dc.GetNextDocument(doc)
Wend
```

ED #6: Wizards

- A Wizard interface in Notes Client:
 - ♦ Create a form with a tabbed table
 - ♦ Set the tabs to "1", "2", "3", etc.
 - ♦ Select "Switch Rows Programmatically"
 - ♦ Set the "name" field to the name of the table
 - ▶ e.g., "RequestTable"
 - ♦ Create a variable on the form with \$Name
 - ▶ e.g., "\$RequestTable"
 - ♦ Have "forward" and "back" buttons increment/decrement the variable

ED #7: Consuming Web Services

- Two approaches
 - ♦ Quick and dirty
 - ▶ Install MS Soap on client machines
 - ▶ Write LotusScript to create an MS Soap Object
 - *Pro: Very quick, handy for testing*
 - *Con: Platform specific, requires DLL on clients, no timeout*



```
Dim Client As Variant
Set Client = CreateObject("MSSOAP.SoapClient")

'Initialize connection to the Web Service
Call Client.mssoapinit ("http://localhost/testWS.nsf/Simple?wsdl")

'Call our simple GetEmailAddress function provided by Web service
Dim result As String
result = Client.getJoke()

'output result to message box
MessageBox result, 48, "Get Joke"
```

ED #7: Consuming Web Services (cont.)

- **Second Approach**

- ♦ **Big and robust**

- ▶ Create a Java Agent that you pass a Notes document to – with all your setup information
- ▶ Read the document in your Java Agent, and consume the Web service
- ▶ Write the results back to your Notes document
- ▶ Once the Java Agent returns, read the document for the results

- ♦ **Pro:**

- ▶ Multi-platform, scalable

- ♦ **Con**

- ▶ Quite a lot of work

ED #8: Classes

- What are Classes?

- ◆ Classes help to:
 - ▶ Bundle data and code in one place
 - ▶ Decompose problems into “objects”
 - ▶ Write smaller, more focused code
 - ▶ Define and implement the internal data model
- ◆ Classes aid reusability
- ◆ Pros: Good design methodology, leads to Java
- ◆ Cons: “Different,” and takes time to sink in



ED #8: Classes (cont.)

Class Person

```
private nnName as NotesName  
private strUNID as String
```

```
sub new(strNewName as string, strNewUNID asString)  
    me.nnName = new NotesName(strNewName)  
    me.strUNID = strNewUNID  
end sub
```

```
public function getName as String  
    if (me.nnName is nothing) then exit function  
    getName = nnName.Canonical  
end function
```

```
public function getUNID as String  
    getUNID = strUNID  
end function  
end class
```

ED #9: Evaluate

- Evaluate allows you to run @Functions within LotusScript
- Sometimes faster, easier
- Example:
`evaluate(|@unique|)`
- DON'T:
 - ◆ Overuse it. Lots of LotusScript functions mimic @functions.
 - ▶ "strRight" == @StrRight

ED #10: "Trusted Servers"

- Scheduled Agents cannot *normally* open databases on other servers

```
not allowed to use mirrors.  
Trusted servers: LocalDomainServers
```

- "Trusted Servers" field in R6 server document, security section, allows servers to "trust" other servers
- This allows you to centralize "collection" Agents
- Caveat: Don't trust servers in another domain!
- **Pros:**
 - Simplifies architecture
 - Fewer Agents
- **Con:**
 - Relies on fast, reliable network infrastructure ...

What We'll Cover ...

- Introduction
- Nursery Slopes
- Everyday Tips
- **Advanced Tips**
- Wrap-up

Advanced #1: Understanding Binding

- There are two types of binding
 - ◆ Early: Set by the compiler
 - ▶ Pros: Type checking, speed, and ease of use
 - ▶ Example: "Dim S as String"
 - ◆ Late: Set at runtime
 - ▶ Pros: Flexibility, NO type checking
 - ▶ Cons: Performance, runtime errors

```
Dim V as variant
Dim S as new NotesSession
set V = S.CurrentDatabase

print V.getTitle()
```

Advanced #2: Coding for Performance

- Remember – Expensive operations include:
 - ♦ Opening databases and views
 - ♦ Documents with lots of fields
- When collecting data:
 - ♦ Cache views where possible
 - ♦ Use NotesViewEntry instead of opening documents
- Example:
 - ♦ 100,000 document database
 - ♦ 7 hours for “open every document in database”
 - ▶ Not using views
 - ♦ 60 minutes using NotesView and opening documents
 - ♦ 12 minutes for “notesviewEntry”



Advanced #3: Lists and Classes

- Classes bind complex data and operations
- List looks these up quickly in memory
 - ♦ Example – Let's extend our Person class:

```
dim People list as Person
dim PeopleByUNID list as Person

Dim P as new Person("Joe Bloggs/ACME", "010101010201020")
....
set People(P.getName) = P
set PeopleByUNID(P.getUNID) = P

if (isElement(People("Joe Bloggs/ACME"))) then _
    Print "Joe's UNID is: " + People("Joe Bloggs/ACME").getUNID

if (isElement(PeopleByUNID("010101010201020"))) then _

    Print "UNID '010101010201020' is: " + _
    PeopleByUNID("010101010201020").getName
```

Advanced #4: Class Inheritance

- Class inheritance allows us to “Extend” classes to add functionality



```
class StaffPerson as Person
  private strStaffID as String

  sub new(strNewPerson as String, strNewUNID as String)
  end sub

  public function setStaffNumber(newNo as String)
    strStaffID = newNo
  end function

  public function getStaffNumber as String
    getStaffNumber = me.strStaffID
  end function
end class
```

Advanced #5: Platform-Specific Code With Classes

```
Dim s as new NotesSession  
Dim mem as variant
```

```
select case s.platform  
  case "Windows/32"  
    set mem = new getMemW32()  
  case "AIX"  
    set mem = new getMemAIX()  
  case else  
    Print "Platform not supported"  
    set mem = nothing  
end case
```

```
if (not mem is nothing) then  
  call mem.printMemory()
```

```
Class getMem  
  function getMem() as long  
    getMem = 0  
  end function  
  sub printMemory  
    print me.getMem()  
  end sub  
end class
```

```
Class getMemW32 as getMem  
  function getMem() as long  
    getMem =  
    getWindowsMemory()  
  end function  
end class
```

```
Class getMemAIX as getMem  
  function getMem() as long  
    getMem = getAIXMemory()  
  end function  
end class
```

Advanced #6: Version-Specific Code With Classes

```
Dim s as new NotesSession  
dim vCU as variant
```

```
select case s.version  
  case 5  
    set vCU = new createUser()  
  case 6  
    set vCU = new createUserv6()  
  case else  
    Print "Version not supported"  
    set vCU = nothing  
end case
```

```
if (not vCU is nothing) then _  
  call vCU.CreateUser(....)
```

Class createUser
function createUser(...) as integer
....
end function
end class

Class createUserv6 as createUser
function createUser(...) as integer
....
end function
end class

Advanced #7: LSI_Info()/GetThreadInfo

- **LSI_INFO()** gives some runtime information
- **Superceded by GetThreadInfo**
 - ◆ GetThreadInfo(11) gives calling class
 - ◆ GetThreadInfo(10) gives function name
 - ◆ and lots more ...
- **Why?**
 - ◆ Error trapping: We can track where we came from
 - ◆ We don't have to pass lists of parameters to error trapping code
 - ◆ Prevents "cut-n-paste coding" errors ...

Advanced #7: LSI_Info()/GetThreadInfo (cont.)

```

                                ' calling code...
                                ...
                                ExitFunction:
                                    exit function
                                errorhandler:
                                    Call RaiseError()
                                    resume exitFunction
                                end function

Function RaiseError()
    Dim thisType As String
    Dim es as String
    thisType = Typename(Me)

    ' Not a class, use the calling module instead
    If (thisType = "") Then    thisType = Getthreadinfo(11)

    es = thisType & "::" & Getthreadinfo(10) & ": "

    If (Err = 0) Then
        es = es + "Manually raised an error"
    Else
        es = es + "Run time error: (" + Trim(Str(Err)) + ") " + _
            Error$ + " at line: "+ Trim(Str(Erl))
    End If

    Print es
end function
```

Advanced #8: Execute

- Execute runs LotusScript code from a String

- Example:

```
Dim executeString as String
executeString = |
    print "Hello world"
    dim s as new NotesSession
    dim db as NotesDatabase
    set db = s.currentDatabase
    print "Current Database name is: " + db.Title
|

execute (executeString)
```

- Why?

- Accomodates version/platform differences at runtime

Advanced #9: Advanced Logging

- **OpenNtf “OpenLog” solution:**
 - ♦ Simple script library addition to your code
 - ♦ Provides “called from”, “error”, and “Line number” functionality
 - ♦ Simple to implement
- **Example of our “NSD” system:**
 - ♦ On error trap
 - ♦ Displays all objects in memory

Advanced #10: Mixing Java and LotusScript

- Use each language to its strengths
 - ♦ Java – good for Web services, network I/O, multithreaded operations
 - ♦ LotusScript – traditional Notes development language, works in UI
- How to mix:
 - ♦ Simple: Call an Agent, passing a Notes document
 - ▶ Fast. I didn't say you had to save the document!
 - ▶ Works both ways
 - ♦ LS2J
 - ▶ Call Java from LotusScript
 - ▶ Example on the next slide ...

Advanced #10: Mixing Java and LotusScript ...

```
Option Public
```

```
Use "xlib"
```

```
Uselsx "*javacon"
```

```
Sub Initialize
```

```
Dim mySession As JavaSession
```

```
Dim myClass As JavaClass
```

```
Dim calculator As JavaObject
```

```
Dim a,b,c As Integer
```

```
Set mySession = New JavaSession()
```

```
Set myClass = mySession.GetClass("calculator")
```

```
Set calculator = myClass.CreateObject()
```

```
a = 10
```

```
b = 5
```

```
c = calculator.mul(a,b)
```

```
MessageBox "a * b = " & c
```

```
End Sub
```

```
// Create a Script Library of type "Java" called xlib
// containing the following function:
public class calculator {
    public int add(int a, int b) { return a + b; }
    public int div(int a, int b) { return a / b; }
    public int mul(int a, int b) { return a * b; }
    public int sub(int a, int b) { return a - b; }
}
```

What We'll Cover ...

- Introduction
- Nursery Slopes
- Everyday Tips
- Advanced Tips
- **Wrap-up**

Resources

- Homework:

- Classes – My OO presentation

- ▶ <http://www.billbuchan.com/web.nsf/htdocs/BBUN6MQECQ.htm>



Resource

- Steve McConnell, *Code Complete 2* (MS Press, 2004).

- www.amazon.com/gp/product/0735619670

- Duffbert's presentation on Java

- <http://hostit1.connectria.com/twduff/home.nsf/htdocs/TDUF5VAQSY.htm>

- Bob Balaban, *Programming Domino 4.6 with Java* (M&T Books, 1998).

- www.amazon.com/gp/product/1558515836

Resources (cont.)

- **LS 2004/AD104 – LotusScript Tips & Tricks**
 - ♦ www-10.lotus.com/ldd/sandbox.nsf
- **SearchDomino.com – LotusScript Learning Guide**
 - ♦ http://searchdomino.techtarget.com/originalContent/0,289142,sid4_gci1001826,00.html
- **NSFTools – Notes Tips**
 - ♦ www.nsftools.com/tips/NotesTips.htm
- **The Notes FAQ !**
 - ♦ www.keysolutions.com/NotesFAQ/



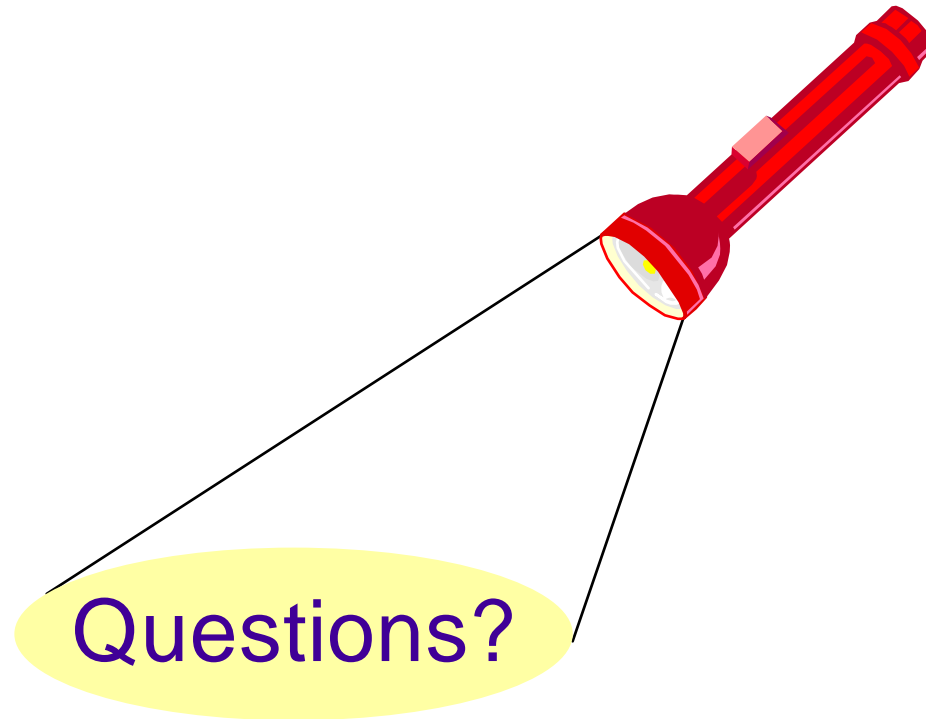
Resources (cont.)

- Brian Benz and Rocky Oliver, *Lotus Notes and Domino 6 Programming Bible* (Wiley 2003).
 - ♦ www.amazon.com/gp/product/0764526111
- Notes.Net (of course)
 - ♦ www.notes.net

7 Key Points to Take Home

- You never stop learning good development techniques
- Develop your code for maintenance
- Good architecture is the best starting point
- Understand the problem area before proposing solutions
- Spend a little time each day reading blogs, THE VIEW, and the support database
 - ♦ Even though specific articles may not be of immediate interest, they may pay off in the future
- A problem shared/discussed is a problem better defined
- Have fun and enjoy your work

Your Turn!



**How to Contact Me:
Bill Buchan
Bill@hadsl.com**