Calling the Notes C API from LotusScript

Bill Buchan
HADSL

What We'll Cover …

- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Wrap-up

Target and Purpose

- Who is the target audience?
  - Advanced Lotus Notes Developers
  - Developers who have a little experience in C API
- What is this about?
  - This talk aims to demonstrate advanced LotusScript, showing:
    - The Notes C API interface
    - The LSX interface
    - Pros and cons of each

Notes C API

- Why Notes C API?
  - 900+ Supported Methods
  - Many complex tasks can be called via API
- Who am I?
  - Bill Buchan
  - Dual PCLP in v3, v4, v5, v6, v7
  - Ten+ years senior development consultancy for Enterprise customers
    - Learn from my pain!
  - Five+ years code auditing
  - CEO of HADSL — developing best-practice tools

Notes C API Programming Toolkit

- Where can you find information on the C API?
  - The Lotus Notes C Programming toolkit
- The toolkit contains:
  - Contains two documentation databases
  - Each function and structure is documented
  - A large number of sample C programs are included
  - Compile time libraries for all platforms
- Generally speaking, backward compatibility is good:
  - For instance, if you have v6 and v7 servers, creating the program using v6 of the API toolkit means that it will run on v6 and v7 servers

Note

What We'll Cover …

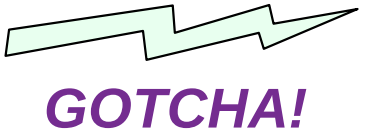
- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Wrap-up

Notes/Domino Architectures

- Domino is a multi-platform server product
- Notes now supports four clients:
  - Windows, Mac (Power PC), Mac (Intel), and Linux
- LotusScript supported from v4.x of Notes, on all server and client platforms
  - Rich set of APIs for general business logic
  - Robust, supported environment for general business applications
  - Amazing level of multi-platform support
- What if you want to go further?
  - Exploit a piece of Notes C API interface?
  - Call a platform specific DLL?

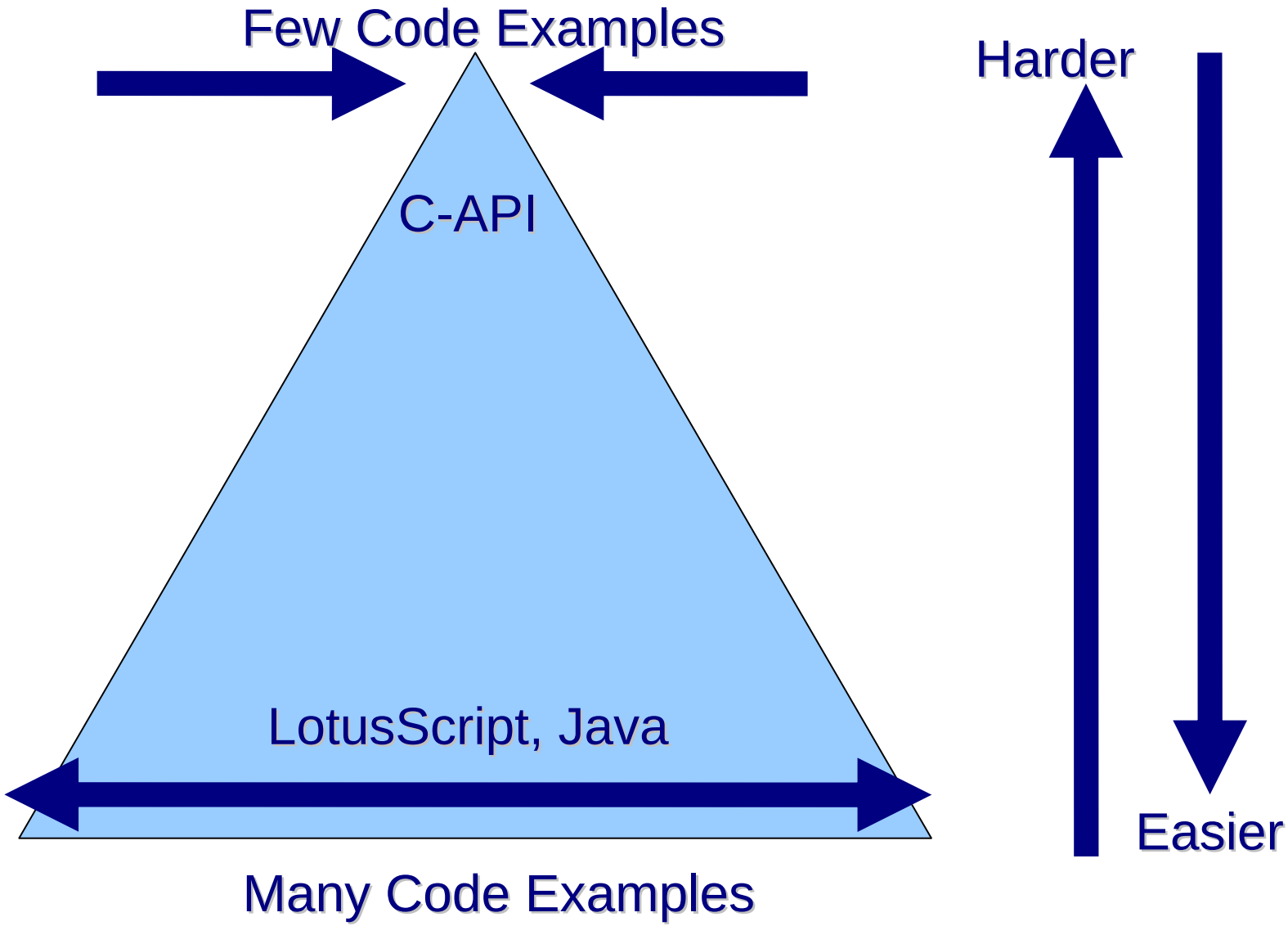
But Surely

- Currently, something like:
  - 90% of all Domino servers run on Windows
  - 95% of all Notes clients run on Windows
- Why not write platform code for that and that alone?
  - A short sighted decision. Your environment may start supporting more platforms.
  - Windows 64-bit coming over the horizon
    - Treat it as a separate platform
      - *As was 16-bit Windows*



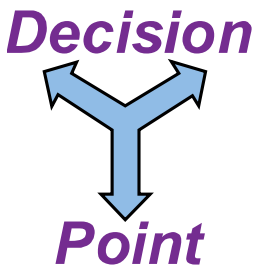
- Corporate push to centralize and consolidate
  - Especially on medium-high level platforms
    - Linux, Solaris, AIX, HP/UX

Be Pragmatic — The Skills Triangle



Be Pragmatic

- Call the Notes C API from LotusScript if:
  - You cannot find a supported architectural solution
  - You have the skills to support LSX and/or Notes C API coding
  - You might possibly require multi-platform code
  - You accept that this code will be harder to write, harder to maintain, and harder to support



Architectures — Conclusion

- If you have to step outside the comfort zone, you have two choices:
  - LotusScript eXtension Toolkit — LSX
    - This builds a library — on windows, a DLL — which allows you to extend LotusScript with custom C code
    - Fairly straightforward
    - A different version for each platform
  - LotusScript calling the Notes C API interface directly
    - Not straightforward
    - One code base can cover all platforms

What We'll Cover …

- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Pros and cons

What Is the LSX Toolkit?

- A C-based API for Lotus Notes
- Interfaces with LotusScript
- Allows you to build platform-specific libraries
- Pros
  - Straightforward for a competent C Programmer
- Cons
  - Platform — and sometimes version(!) — specific
    - You have to produce a version (from the same source code) for each platform
  - Lack of support
    - Last release — March 2001
    - Visual Studio 2005 not supported

Note

What Can You Use the LSX Toolkit for?

- Used to use it for Notes Database work
  - We have a Notes Database installer which unpacks, design refreshes, sets ACLs, etc.
  - Initially coded that as LSX
  - Now use LotusScript calls to Notes C API directly
    - Deployment of LSXs is non-trivial
    - Especially updating LSXs that are in use!
- Still use it for Active Directory Integration
  - AD integration possible using LotusScript/COM interface — but unreliable
  - Coded Active Directory Services Interface (ADSI) to manage objects within AD — from within LotusScript!

How Can You Create an LSX With the Toolkit?

- Design a data structure you wish to expose in LotusScript
- Use the LSX Wizard to create a relevant LSX project
- Use a supported C-Compiler for your platform(s) to create code to fill in the blanks
- Run Do_It to compile your library
  - Windows produces a .DLL file
  - *nix produces a .so file
- Place that library in the executable code directory of your server and/or client
- Test, test, test! 16-bit

Checklist

LSX Wizard (1)

- Create a new Project

LSX Wizard (2)

- Now add a class to the project

LSX Wizard (3)

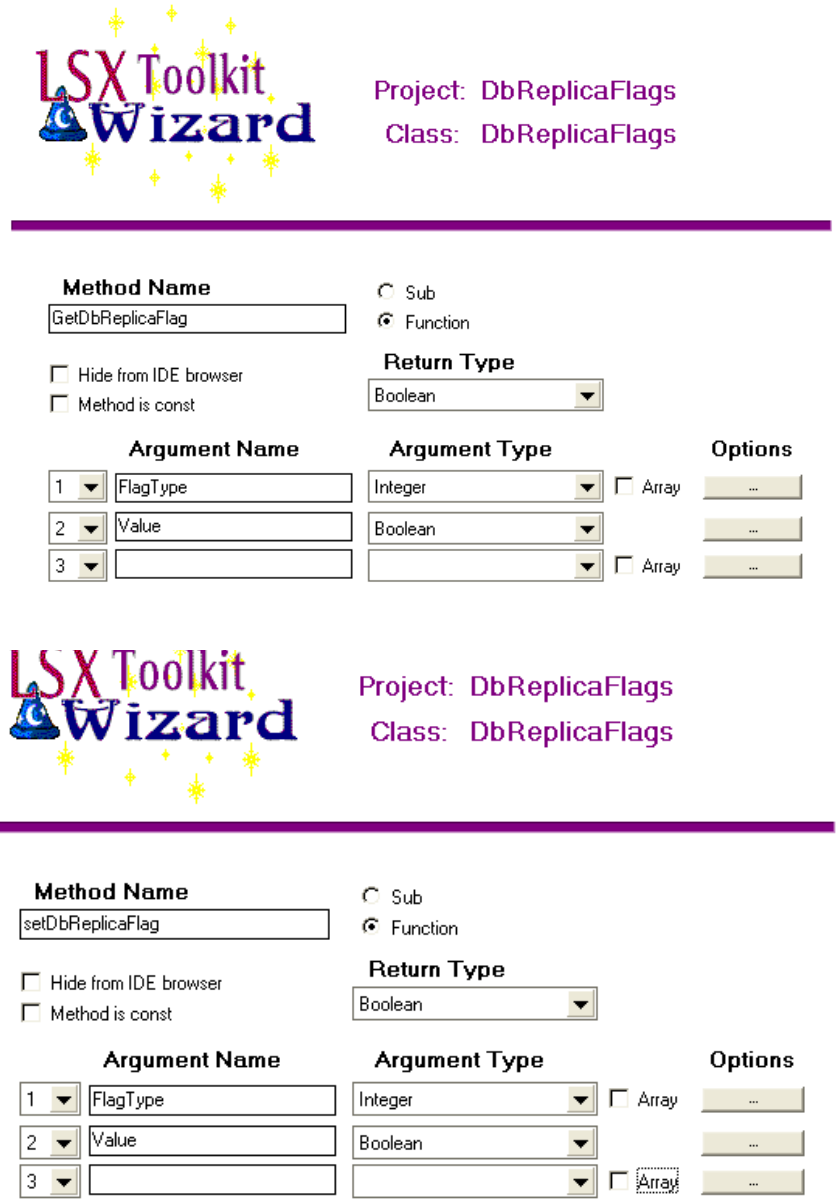
- Override the Constructor

LSX Wizard (4)

- Add a “Close” Sub

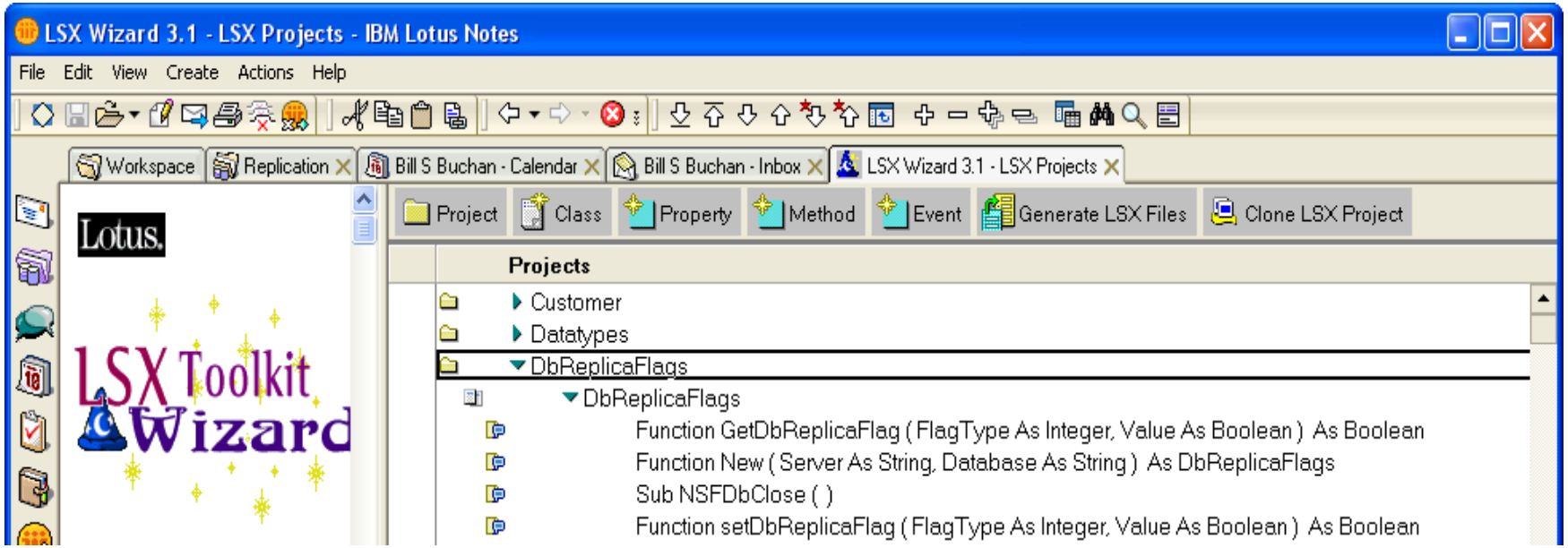
LSX Wizard (5)

- Add a Getter and Setter



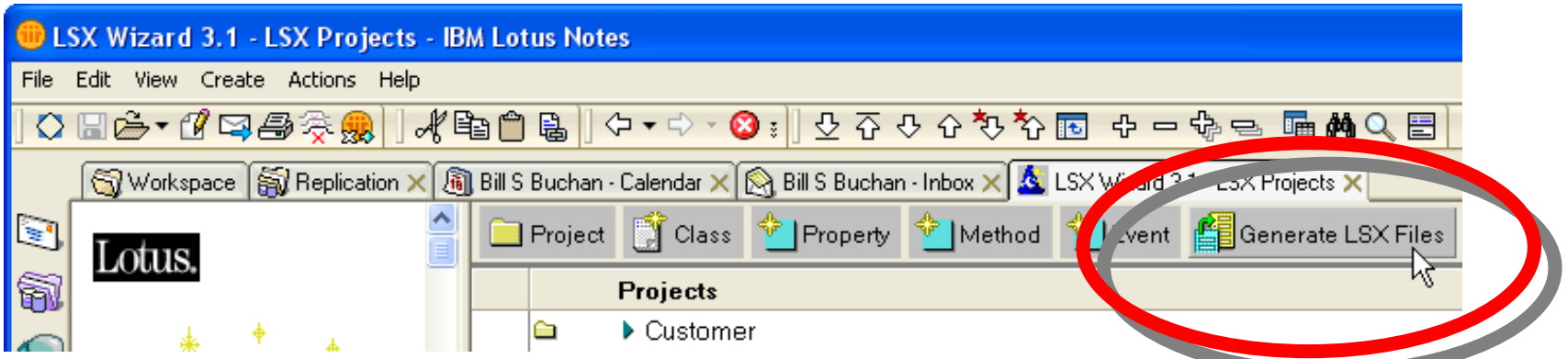
LSX Wizard (6)

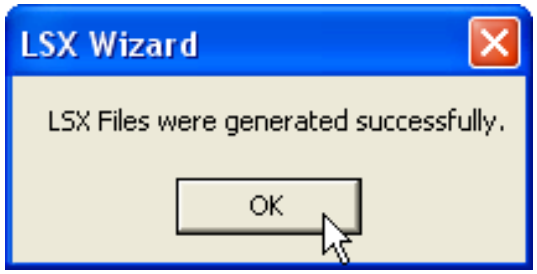
- You should now see this:



LSX Wizard (7)

- Now generate code





LSX Wizard (8)

- Now fill in the code in the LSX source files

```
//{{LSX_AUTHOR_CODE_Include_1
//}}

#include "DbReplicaFlags.hpp"
// includes for the objects defined in your LSX
//{{LSX_AUTHOR_CODE_Include_2
//}}
```

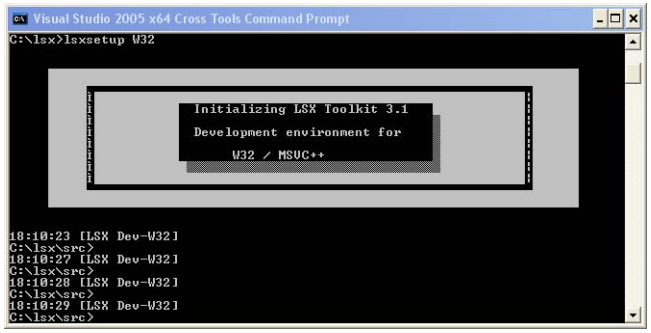
LSX Wizard (9)

- Now fill in the code in the LSX source files

```
DbReplicaFlags:: DbReplicaFlags(
   LSPTR(LSXLsiSession) pContainer, LSXString&
   Server, LSXString& Database)
    : LSXBase((LSPLTSTR)"DbReplicaFlags",
   pContainer->LSXGetInstance(),
       CDBREPLICAFLAGS_DBREPLICAFLAGS_ID,
         pContainer)

   //{{LSX_AUTHOR_CODE_Additional_Base_Class_Init1
    //}}

    //{{LSX_AUTHOR_CODE_Internal_Member_Init1
    //}}
{
    //{{LSX_AUTHOR_CODE_Constructor1
    //}}

}
```

LSX Wizard (10)

- Run a command prompt
  - Go to the LSX directory
  - Run **`LSXSetup W32`**
  - Run **`cd DbReplicaFlags`**
  - Run **`Do_it`**
    - This will compile the DLL file
  - Copy the DLL from \lsx\bin\w32\DbReplicaFlags.dll to your Notes Program Directory
- Note:
  - The Notes client (or server) usually holds onto the DLL once loaded, so refreshing the LSX usually requires a client (or server) restart



Heads Up!

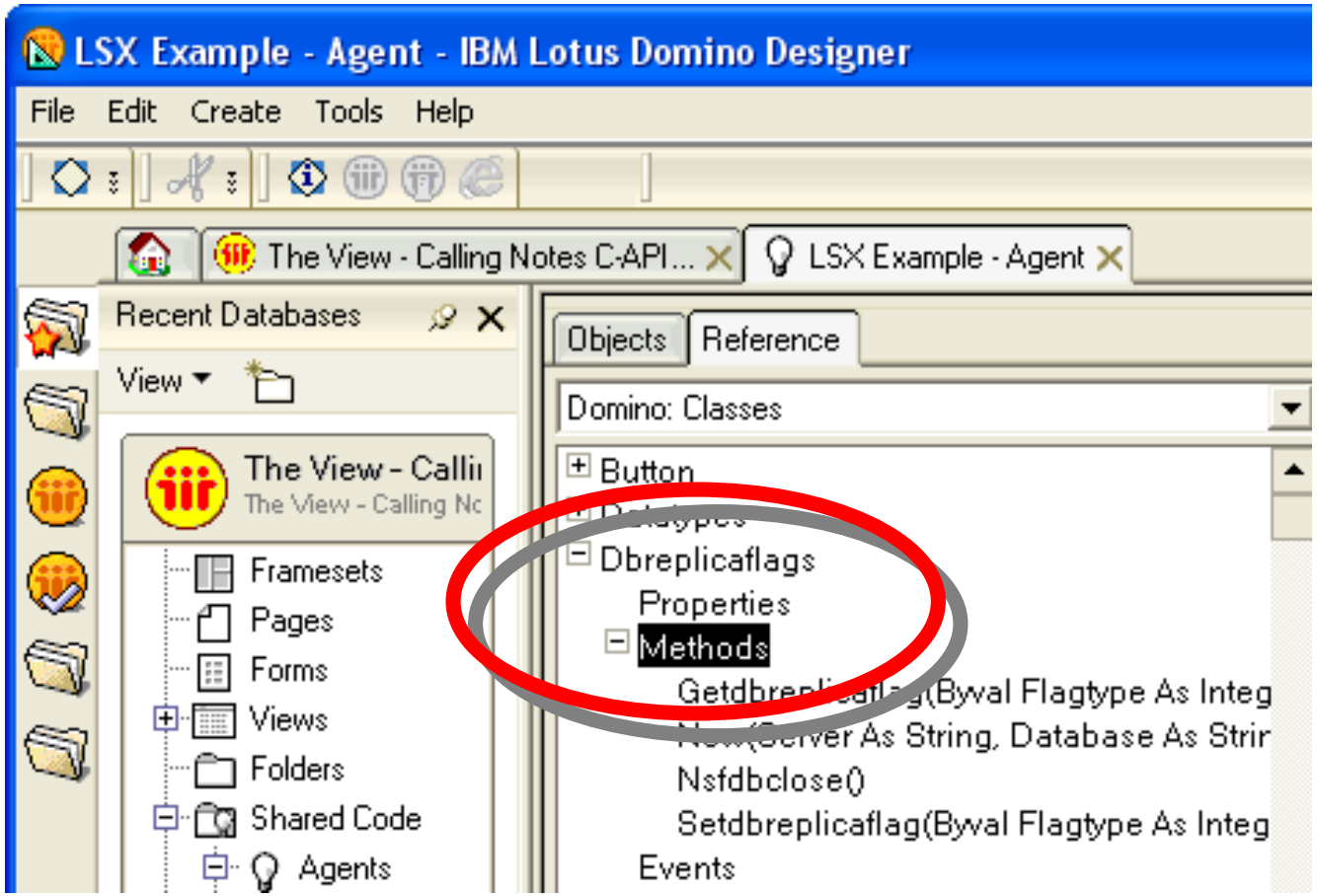
LSX Wizard (11)

- Create a LotusScript Agent

```
Uselsx “DbReplicaFlags.dll”          ' in 'Options'

Sub Initialize
    Dim DR As New DbReplicaFlags("", "names.nsf")
    Dim value As Integer

    If dr.GetDbReplicaFlag(REPLFLG_DISABLE, value) Then

        If (value) Then
            Print "Replica Disable is set to 'TRUE'"
        Else
            Print "Replica Disable is set to 'False'"
        End If
    Else
        Print "I failed to get the replica flag "
    End If

    Call dr.NSFDbClose()
End Sub
```

LSX Wizard (12)

- As a side effect of loading the LSX
  - The LSX signature is now available in the designer



Demo

Demo

Overview
the LSX Toolkit

LSX Summary

- LSX Toolkit is fairly straightforward
- Since the LSX Toolkit supports multiple platforms and the same source code is used on each:
  - You do have to recompile the LSX on multiple platforms
  - You don't have to take account of many platform differences
- It's really relevant for extending your Notes/Domino code to new APIs
  - e.g., You can call Active Directory API calls in order to integrate Domino with Active Directory

Checklist

What We'll Cover …

- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Wrap-up

Platform Differences: Definitions

- In normal LotusScript, platform differences are taken care of
  - Even in LSX programming most differences are taken care of
  - Not so in direct LotusScript to C API calls
- Each platform represents data in a subtly different manner
  - For instance, the HANDLE datatype has different lengths on different platforms:
    - 32 bits long in Windows and OS/400
    - 16 bits long on AIX, Macintosh, etc.
  - Memory alignment widths on iSeries are different from other platforms

Platform Specific implementations

- In order to call the Notes library, you will have to link to different library files on each platform:
  - Windows/32: nnotes.dll
  - Solaris, Linux: libnotes.so
  - AIX: lnotes_r.a
  - HPUX: libnotes.sl
  - MacOs, OS/X: NotesLib
  - OS/400: libnotes.srvpgm
  - OS/390 libnotes
  - OS/2: lnotes.dll
  - Windows Alpha: anotes.dll

Platform Differences: Endians

- Endians
  - Some platforms represent multi-byte numbers with the lowest value bytes in the lower segments of memory — little endians
  - Macintosh (on PowerPC!) represents the larger value bytes in the lower segments of memory — big endians

Coding for Platform Differences

- Our challenge is to construct a code sequence that:
  - Knows which platform it's running on
  - Makes calls to the platform-specific library file
  - Understands platform differences in terms of alignment and data item size
  - And most importantly — fails "safe" when presented with a new platform that it cannot deal with
- This is not a trivial exercise
  - My estimate is that this exercise will take at least 5-10 times more effort — development + testing — than a normal LotusScript business function

Heads Up!

Decoding Signatures

- A “Signature” is:
  - The definition of a function calls' parameters and return value
  - These can be found in the Notes “C API 7.0 Reference for Domino and Notes”, at www-10.lotus.com/ldd/notesua.nsf/0b345eb9d127270b8525665d006bc355/6a92a658297e6baf852571a3007143be?OpenDocument
    - Example signature:

```
STATUS LNPUBLIC NSFNoteLSCompile(
DBHANDLE  hDb, NOTEHANDLE  hNote, DWORD  dwFlags);
```

    - In Windows:

```
Declare Function W32_NSFLSCompile Lib LIB_W32
Alias "NSFNoteLSCompile"
ByVal hdb As  Long,
ByVal hNote As  Long,
ByVal null1 As  Long )
As  Integer
```

Decoding Signatures (cont.)

C-API	Win32	Linux	AIX	Solaris	Mac
BYTE	BYTE	BYTE	BYTE	BYTE	BYTE
BOOL	Long	Long	Long	Long	Integer
Int	Long	Long	Long	Long	Long
Long Int	Long	Long	Long	Long	Long
WORD	Integer	Integer	Integer	Integer	Integer
SWORD	Integer	Integer	Integer	Integer	Integer
DWORD	Long	Long	Long	Long	Long
LONG Int	Long	Long	Long	Long	Long
HANDLE	Long	Integer	Integer	Integer	Integer
NOTEHANDLE	Long	Integer	Integer	Integer	Integer
DBHANDLE	Long	Integer	Integer	Integer	Integer
MEMHANDLE	Long	Long	Long	Long	Long
STATUS	Integer	Integer	Integer	Integer	Integer
Char *	String	String	String	String	String

What We'll Cover …

- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Wrap-up

Calling a Simple C API Function

- Let's get our feet wet with some simple API
  - No complex sequence of calls handling a shared memory resource
    - A single call with simple datatypes
  - A call that will not result in client memory corruption should we get it wrong. Hopefully.
- Let's call it from Windows and ignore other platforms
  - A function that tells you the network latency — in milliseconds — between the current Notes instance and a target server
    - **`NSFGetServerLatency`**

NSFGetServerLatency() – API Reference

- We shall call:
  - NSFGetServerLatency()
  - This returns the network latency time described in milliseconds for a call to a remote server
  - Useful to decide which server is closest in terms of network response
  - Its signature from the C API reference is:

```
STATUS LNPUBLIC NSFGetServerLatency(
    char far *ServerName,
    DWORD  Timeout,
    DWORD far *retClientToServerMS,
    DWORD far *retServerToClientMS,
    WORD far *ServerVersion);
```

NSFGetServerLatency() – API Reference (cont.)

- From the C API Reference:
  - Input Parameters
    - ServerName — Null-terminated string containing the name of the server to query
    - Timeout — Number of milliseconds to wait for a reply from the server. A timeout of 0 indicates that the default timeout value is to be used.
  - Output Parameters
    - (routine) — Return status from this call:
      - *NOERROR — Success*
    - retClientToServerMS — Optional — If not NULL, the number of milliseconds required to send the request to the server is stored at this address
    - retServerToClientMS — Optional — If not NULL, the number of milliseconds required for the reply to return from the server is stored at this address
    - ServerVersion — Optional — If not NULL, the server version (the Domino build number for the server) is stored at this address

Simple C API Calling

```
' This is a constant for our windows-based
' Library file:
Const LIB_W32 = "nnotes.dll"

' Declare our function for windows
Declare Function W32_NSFGetServerLatency _
   Lib LIB_W32 Alias {NSFGetServerLatency} (_
   Byval ServerName As Lmbcs String, _
   Byval Timeout As Long, _
   retClientToServerMS As Long, _
   retServerToClientMS As Long, _
   ServerVersion As Integer) As Integer
```

Simple C API Calling: Execution

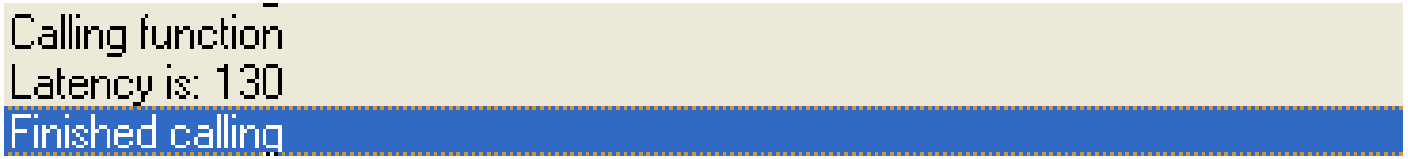
```
' A function to get network latency time...
Public Function getServerLatency _
(strServer As String) As Long
   Dim nnServer As New NotesName(strServer)
   Dim ToServer As Long, fromServer As Long
   Dim ver As Integer
   Dim timeout As Long

   timeout = 1000       ' 1000ms == 1 second

   Call W32_NSFGetServerLatency(nnServer.Canonical,_
     timeout, toServer, fromServer, ver)

   ' Return both directional latencies added together
   getServerLatency = fromServer + ToServer
End Function
```

Simple C API Calling: Execution (cont.)

```
Sub initialise
   Print “Calling function”
   Print “Latency is: “ + _
      cstr(getServerLatency(“domino-
        90.hadsl.com/HADSL/US”))
   Print “Finished calling”
end sub
```

Simple C API Calling: The Results

- Running the agent “Example1” in the database produces the following runtime output:



- Now — this is not production code! It requires:
  - Error handling
  - Multi-platform support

What Can't You Call?

- Callback routines
  - Some Notes C API functions require you to specify mandatory callback routines — other pieces of C API that will get called by the target C API routine

    - LotusScript does not (and IMHO never will) support this
    - This rules out Extension Manager Routines, Menu Addin Routines
  - In terms of callback routines that allow you to specify optional callbacks (progress status indicators, etc), you can pass in NULL (or ZERO), and the callback function will be ignored

What We'll Cover …

- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Wrap-up

Defining Complex C API

- Complex C API is where:
  - More than one function call is required to perform a task
    - and/or
  - A HANDLE of some description is needed to reference a data object

API Handles

- A handle is a numeric reference to a structure in memory that the C API interface has created
  - For example, a handle to a Notes Database
- You do not deal with the memory structure itself — you deal with the handle to the memory structure
  - Example: **`NSFDbOpen()`** creates a new memory structure and gives you back the handle to that
  - You perform some work using **`NSFDInfoGet()`**
  - You close the database using **`NSFDbClose()`**

Some Rules on Handles

- There are different types of handles:
  - Document handle, database handle, etc.
  - Do not mix these handles up!
    - Doing so will crash the session/client/server
- Always properly close your handles
  - You have to properly trap all code branches
  - You have to keep track of the handle until you specifically de-allocate it
    - Not doing so will crash the session/client/server
    - It may not crash immediately. It may crash when the session, client or server closes.
- You cannot change the value of the handle
  - Doing so will crash the session/client/server

Coding Around Handles

- A really good way is to use a class
  - It performs the “open” part of the handle operation on the class constructor
  - It performs the “close” part of the handle operation on the class destructor
    - This means that no matter what happens, the handle operation will be properly closed even if this is not specifically coded in your LotusScript
  - All the code specific to this handle operation is embedded in the class
    - And can include multi-platform code
  - All you see in your LotusScript is an operation with this specific class

Database Handle Class – Introduction

Demo

Demo of
DbHandleManager

Handle Summary

- If you get a handle, you have to de-allocate it
- Only use handle data you know is valid
  - If a handle == NULLHANDLE (or ZERO), then it's not a valid handle
- Bonus:
  - NotesDocument.handle returns the current handle
  - You can easily use LotusScript to:
    - find a document
    - use direct calls to update information that the object model does not support (at the moment)
    - e.g., Writing Notes Replica Items

Secret

Complex C API

- Where more than one function call is required to perform a function, it is recommended that:
  - You use defensive coding techniques to understand and make “safe” every single coding branch possible
  - Any handle information used during this sequence is kept safe and its de-allocation specifically catered for
  - You use classes to ensure proper construction and destruction of handle information

What We'll Cover …

- Introduction
- Architectures
- The LotusScript eXtension toolkit
- Platform differences
- Calling simple Notes C API
- Complex Notes C API
- Wrap-up

Know Your Battleground

- LotusScript coding is:
  - Fairly straightforward
  - Supported
  - Platform independent
- Stepping outside of this comfort zone is going to be hard
  - Take far more time in development and testing
  - Push development resources to the limit
  - Only do this if you have an absolute business need

Heads Up!

LSX Versus Notes C API

- You need to establish, based on your requirements, which is the best fit:

Feature	LSX	LS calling Notes C-API directory
Deployment	Requires code deployment	No Deployment required
Multi-platform	Different LSX for each platform	Same code in Library
C-API integration	Very good integration – callbacks, etc	Basic integration
Code Difficulty	Medium	Hard
Stability	Extremely Stable	Stable

Checklist

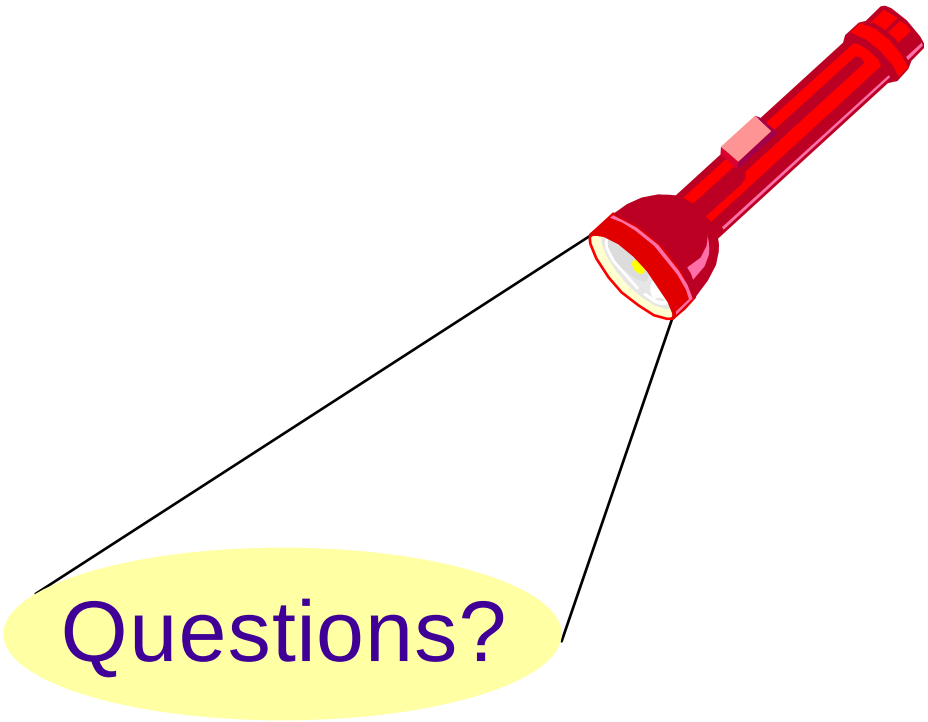
Resources

- Normunds Kalnberzin, “LotusScript to Lotus C API Programming Guide,” www.Ls2Capi.com
  - The ultimate reference
  - Ebook for €18, Book + shipping for €42
  - It pays for itself in a single hour
- Notes C API Reference, LSX Toolkit
  - http://www-128.ibm.com/developerworks/lotus/downloads/toolkits.html
- Julian Robichaux, “Notes API Tips,” www.nsftools.com/tips/APITips.htm
  - Lotus Notes API tips

7 Key Points to Take Home

- Calling Notes C API is hard
  - It has to be tested on all platforms
  - It can easily take out a client or server process
  - Only use it if absolutely required
  - Use defensive coding!
  - It's the next stage beyond the well-regulated LotusScript sandbox
  - It might be simpler to use LSX than Direct LotusScript to C API calls
    - You certainly get more control and debugging
  - It gives you all possible Notes capability

Your Turn!



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