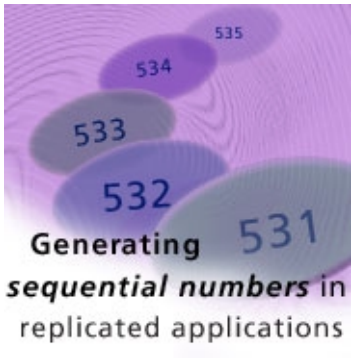
Notes.net

Iris Today



by
Jonathan
Coombs

Level: Intermediate
Works with: Domino 5.0
Updated: 02/04/2002

One of Notes/Domino's greatest strengths is its ability to keep multiple database replicas synchronized across multiple servers and remote clients. However, this synchronization is not instantaneous, so frequent updates to the same document across different replicas often generate replication conflicts. Because of this risk, it's important to design any replicated database in a way that encourages a given document's group of editors to all edit it on the same replica. Furthermore, if the application itself updates any system documents, there should only be one server on which it updates a given document.

One practical implication of this limitation is that replicated applications can't generate sequential numbers very easily. It's fairly simple to build a sequential number generator that accesses and increments a hidden counter document in the current database. But if that database is then replicated across several servers and remote laptops, how is each replica to instantly update the sequential counter? If a database with sequential numbering does not take these issues into account, it cannot be successfully replicated, because each replica would attempt to maintain its own counter, resulting in duplicate numbers and replication conflicts on the counter document.

It is also a good idea to take replication into account when you do not yet intend for your application to be replicated. Even if you can guarantee that it will not be replicated across servers, it's harder to prevent individual users from creating local replicas. And of course, it's not uncommon for a successful application to start out with a small local user base and then be expanded to include a large international user base.

This article presents a reusable AutoNumbering tool that supports replication by designating one "central" replica as the source of all sequential numbers. All documents created in this central replica are numbered immediately, but all other documents are numbered by a scheduled agent once they arrive in the central replica. This article also briefly explores some other replication-safe alternatives.

The reusable AutoNumbering tool is available in the **sample AutoNumbering database** in the Iris Sandbox, along with some sample design elements to illustrate its functionality. This kind of tool can be used for robust incremental numbering in both replicated and nonreplicated applications.

Preparing an application for sequential numbering

There are many kinds of applications that use sequential numbers. An order tracking database is a typical example, so I have included a very basic order form as the example in the sample database. To keep it simple, I've assumed that customers will manually enter all of the order information (item, quantity, and special instructions).

Example: an order tracking database

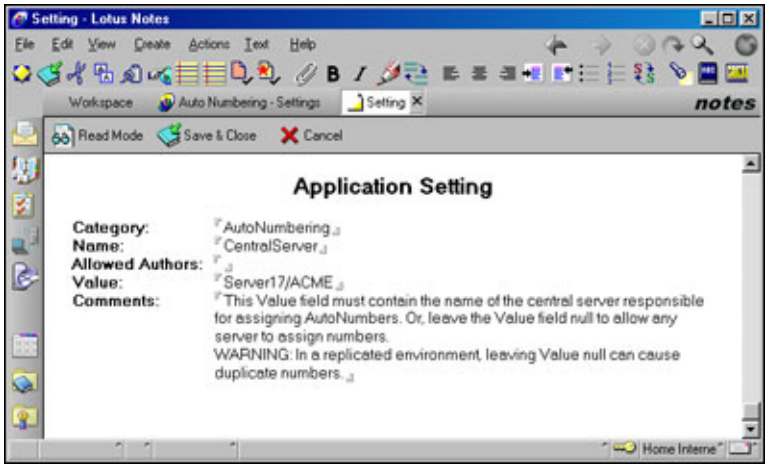
If you look at this form in Designer, you will see that each field name is prefixed with *f* or *fd*: fDocNumber, fdDocNumber, fCustomer, fItem, fQty, fComments, and fHardCode. These prefixes mean *field* and *display field*, respectively, and conform to the naming conventions I use throughout the database's design. The fDocNumber field is really the only significant field on the sample form, because it will store the sequential numbers. The field name is completely arbitrary, since it will be passed to the number generator as a parameter. Since the field should not be edited, it's set to be computed-when-composed.

Three necessary settings

There are three pieces of information that need to be provided by the application in order for the AutoNumbering tool to function properly. The first two are the name of the central server and the number of digits desired per number. These two parameters can either be stored along with the application's other settings, or they can be hard-coded.

The third necessary setting is the sequential number counter. It cannot be hard-coded because its value will be incremented with each new order. To keep things simple, the AutoNumbering code assumes that this counter will be a numeric field in a normal Notes document (not a profile document) that it can access and modify using a Lookup view.

For the sample application, I installed a generic Application Settings tool and used it to store all three settings. The tool consists of one form (frmSetting) and two views (vwSettings and vwLookSettings). Each setting is stored in its own setting document so that it can be individually maintained and secured.



For more information about the Application Settings tool, refer to the *Iris Today* article, "**Application settings tool: an alternative to profiles**."

Creating the core AutoNumbering features
Now that the sample application and necessary settings are in place, along with appropriate event handlers for requesting new numbers, let's see how to build the actual AutoNumbering functionality.

There are two core functions that can be built once and directly reused in different applications. The first is the function that increments the counter document and returns a unique sequential number. The second is useful in applications that store numbers as fixed-length strings; it converts the sequential number into a string and adds leading zeroes as necessary.

These two functions, AutoNumberGet and AutoNumberCstr, reside in the slANAutoNumbering script library in the **sample database**. To test them directly, you can create an order, check "Use simple hard code," and click Save & Close. The form's QueryClose event will call the AutoNumbering functions using a set of hard-coded parameters:

```
Dim iTemp As Long
iTemp = AutoNumberGet("vwLookSettings",
"AutoNumbering*OrderCounter", "fValue", "", True)
doc.fDocNumber = AutoNumberCstr(iTemp, 5)
Call doc.save (False, False)
```

In plain English, the call to AutoNumberGet says, "Find a document named AutoNumbering*OrderCounter in the vwLookSettings view and get the next order number from its fValue field. Regardless of what server I'm currently on, return the number, increment it by one, and save it back to the counter document." The call to AutoNumberCstr says, "Take the sequential number and add as many leading zeroes as it takes to make it into a string of at least five digits."

Getting a sequential number
So what actually happens inside of AutoNumberGet? Most of the time, not much. Without its error handling code, the core of the function looks something like this:

```
Function AutoNumberGet (strANView As String, strANKey As String,
strANField As String) As Long

    Dim s As New NotesSession, db As NotesDatabase
    Dim vwAN As NotesView, docANCounter As NotesDocument
    Dim varANValue As Variant, lngANValue As Long
```



```
    Dim itemNewAN As NotesItem, iReturn As Long

    Set db = s.CurrentDatabase
    Set vwAN = db.GetView(strANView)

    Set docANCounter = vwAN.GetDocumentByKey (strANKey, True)
    varANValue = docANCounter.GetItemValue( strANField )
    lngANValue = Clng( varANValue(0) )
    Set itemNewAN = docANCounter.ReplaceItemValue _
        ( strANField, Cstr(lngANValue+1) ) 'increment the number by one
    Call docANCounter.save(False,False) 'save the numbering doc
    iReturn = lngANValue

    AutoNumberGet = iReturn
End Function 'AutoNumberGet
```

After declaring the list of parameters and variables, the function finds the specified view, document, and counter field within the current database, and retrieves the counter's value. It then increments and saves the counter so it will be ready for next time. (Notice that the function returns a value of type Long, to support applications with very large sequential numbers.)

As straightforward as this is, there are quite a few situations in which this simple function could fail. Of course, the function must fail gracefully if it's called on any server except the central one, and there are other possible problems too. The counter document might have been deleted, or the user might not have access to modify it, or (in very high-usage applications) there might be a save conflict on the counter document. Finally, some error unforeseen by the programmer might occur.

The rest of the code in the full version of AutoNumberGet is there to protect against these errors. (To examine the full code, see **The AutoNumberGet function** sidebar.) To keep the design simple and protect the calling code, I chose to handle all errors internally and, if necessary, flag them by returning negative numbers. This is simpler than throwing real exceptions, although it is nonstandard and perhaps less flexible.

To prevent replication conflicts, AutoNumberGet takes the name of the central server as a parameter and compares it to the current server, returning -1 if the current server is not the central server. It returns -2 if the counter document doesn't exist, -3 if the current user does not have access to update the counter document, -4 if there is a save conflict, and -9 for other errors.

There are two ways the AutoNumberGet function can handle save conflicts, depending on whether the Fail Proof feature is enabled. If it's enabled and a save conflict occurs, the counter document is released and reopened, and once again a new number is retrieved, incremented, and saved. This process repeats until the counter is successfully incremented. If the Fail Proof feature is disabled, however, a save conflict on the first try causes the function to fail and return -4.

Adding leading zeroes

After a sequential number has been successfully generated, it's often desirable to store it as a fixed-length string. The AutoNumberCstr function takes a positive number, converts it to a string, and determines the difference between its length and the desired length. It then builds a string of zeroes to make up the difference and adds it to the beginning of the numeric string.

```
Function AutoNumberCstr (iAN As Long, iChars As Integer)
As String
```

```
    Dim strANin As String, strReturn As String
    Dim intPadding As Integer, strPadding As String
    Dim iANLength As Integer, i As Integer

    strANin = Cstr (iAN)
    iANLength = Len (strANin)

        If (iChars = 0) Or (iANLength >= iChars) Then
         strReturn = strANin 'no padding needed
    Else
         'Pad with a string of zeroes
         intPadding = iChars-iANLength
         strPadding = ""
         For i = 1 To intPadding
              strPadding = strPadding + "0"
         Next
         strReturn = strPadding + strANin
    End If

End Function 'AutoNumberCstr
```

The slANAutoNumbering script library is intended to be directly reusable across multiple applications without any customization at all. This is important because it allows you to maintain the master copy in a central design template and have each installation inherit its design from that template. Distributing bug fixes and enhancements using design inheritance can be easy and relatively safe, while manual distribution is tedious and prone to introducing discrepancies into the various installed copies over time.

For more information about managing reusable design elements, refer to the **Domino Designer Help** and the *Iris Today* article, "**Application settings tool: an alternative to profiles**."

Connecting an application with the AutoNumbering tool

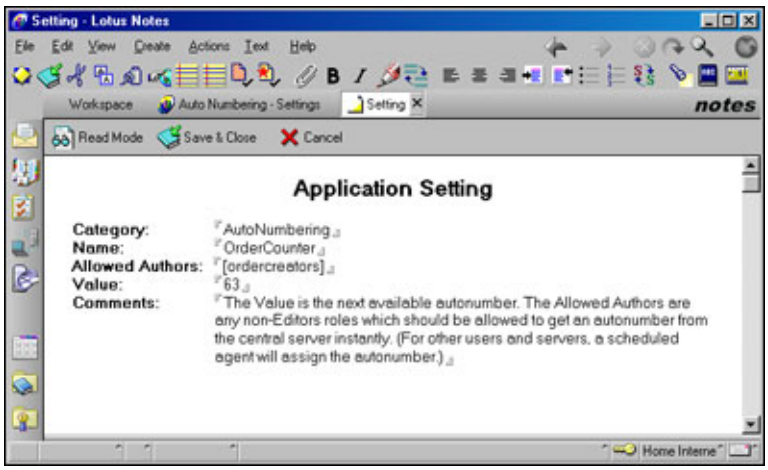
Remember that the number counter must only be modified on one designated "central" server, and so this central server is the only one that can assign new numbers. In the sample application, if an order is created on a noncentral server, the fDocNumber field needs to remain unassigned until the order has replicated to the central server. If the order is created on the central server, however, the application can go ahead and get a number immediately.

Deciding when to assign numbers

We've already seen a simple hard-coded example of requesting a sequential number directly from a Notes client form's QueryClose event. It's also quite likely that the application will be accessed by Web clients. I chose to allow both options, so the form in the sample database directly requests an automatic number both on WebQuerySave and on QueryClose.

Warning: If you choose to allow direct number assignment in a real application, keep in mind that a form's QueryClose code runs in the Notes UI under the current user's authority, and every potential order creator therefore needs permission to update the counter setting. (The scheduled agent and the form's WebQuerySave agent won't have this problem because they run under the authority of the developer who signed them.) So, unfortunately, this convenient and direct approach is not very secure, because any order creator with a Notes client could corrupt the counter by setting it to an invalid number.

As the owner of the order tracking application, I decided to allow immediate number assignment within Notes and trust users not to hack the counter. So, the database's ACL is set to grant Author access and the [ordercreators] role to all Default users, and the OrderCounter setting has included the [ordercreators] role in its Authors field.

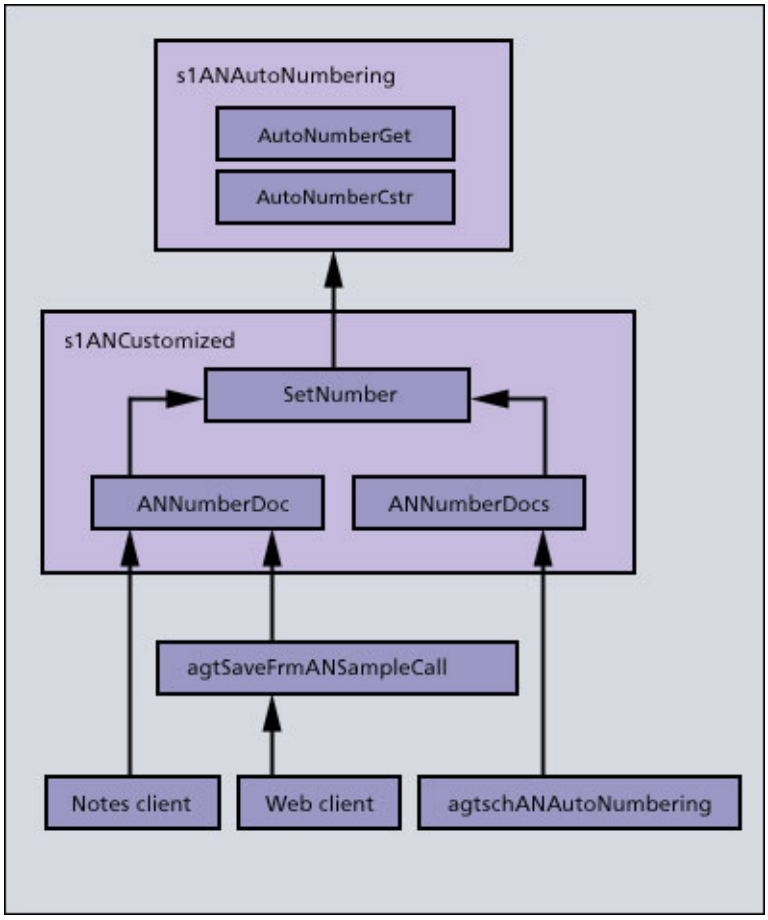


Three paths to the core features

The two core functions, AutoNumberGet and AutoNumberCstr, should be usable by any application without customization. Even so, the application-specific work involved in loading settings and handling error conditions is significant enough that a separate set of partly reusable (customizable) functions are also needed. The **sample database** uses two interface functions, ANNumberDoc and ANNumberDocs, in a script library named slANCustomized. These functions process single requests and batch requests, respectively, and pass them along to a private function named SetNumber. SetNumber handles all interactions with the core AutoNumbering functions.

This approach of separating a set of functionality into generic elements and application-specific interfaces is a good technique for making code more modular and, consequently, more reusable.

As this diagram of function calls shows, there are three paths for requesting a sequential number. Let's examine each path.



To support order submission from a Web client, the sample order form references an agent in its WebQuerySave event. This event executes when a Web user submits the form.

```
@Command([ToolsRunMacro]; "agtSaveFrmANSampleCall")

Use "slANCustomized"
Sub Initialize 'agtSaveFrmANSampleCall
	'Called by frmANSampleCall on web QuerySave
	Dim s As New NotesSession, doc As NotesDocument
	Dim varTemp As Variant
	On Error Goto tagErrorHandler
	Set doc = s.DocumentContext
	Call ANNumberDoc (doc)
	varTemp = doc.GetItemValue(C_AN_NUMBER_FIELD)
	varTemp = Cstr(varTemp(0))
	If ( varTemp <> "") Then
		Print "<p><b>Order Processed. Your order number is " +
		varTemp + ".</b></p>"
	Else
		Print "<p><b>Order Processed." _
		+ " Your order number will be emailed to you.</b></p>"
		'(The email functionality is not included in this sample database.)
	End If
tagEnd:
	Exit Sub
tagErrorHandler:
	Print "<p><b>Error " + Cstr(Err) + ": " + Error$ + "</b></p>"
	Resume tagEnd
```

```
End Sub 'agtSaveFrmANSampleCall
```

This agent consists of little more than a function call wrapped in an error handler. The actual assignment of the sequential number is handled by the ANNumberDoc routine, which will be explained later.

The second path is direct numbering from the Notes client. There is no "submit" event in the Notes client that is equivalent to the WebQuerySave event. Instead, Notes applications commonly use the QuerySave and QueryClose events in combination to detect form submission. The sample order form uses this approach, waiting until the user has both saved and closed before proceeding to request a number:

```
Use "slANCustomized"
Dim bWasSaved As Integer
Sub Postopen(Source As Notesuidocument)
    bWasSaved = False
End Sub
Sub Querysave(Source As Notesuidocument, Continue As Variant)
    bWasSaved = True
End Sub
Sub Queryclose(Source As Notesuidocument, Continue As Variant)
    Dim doc As NotesDocument, iTemp As Long
        If bWasSaved Then
            Set doc = source.Document
            Call ANNumberDoc (doc)
        End If
End Sub
```

The third path applies to all orders submitted on noncentral servers. They eventually replicate to the central server and are handled in batches by the scheduled agent, agtschANAutoNumbering. This agent does nothing more than call the ANNumberDocs routine, which will be explained later.

```
Use "slANCustomized"
Sub Initialize 'agtschANAutoNumbering
    Dim s As New NotesSession, db As NotesDatabase
    Dim dc As NotesDocumentCollection
    Set db = s.CurrentDatabase
    Set dc = db.UnprocessedDocuments
    Call ANNumberDocs (dc)
End Sub 'agtschANAutoNumbering
```

Interfacing with the three paths

One good way of interfacing the three possible execution paths with the core AutoNumbering functions is to create one interface for processing individual documents and one for processing multiple documents. In the sample database, the ANNumberDoc routine supports direct numbering from the Notes and Web clients, and the ANNumberDocs routine supports batch numbering from the scheduled agent.

The ANNumberDoc routine first determines the central server and the number of desired digits, and makes sure that a counter document exists. If the order document in question has not yet been numbered, it requests a new number from SetNumber. (The SetNumber routine will be explained later.)

```
Private Const C_AN_NUMBER_FIELD = "fDocNumber"

Public Sub ANNumberDoc (doc As NotesDocument)
    'Assigns an AutoNumber to a doc if it's on the main server.
    'Called by the document's QueryClose or Web QuerySave event.
```

```
    Dim varTemp As Variant
    Dim strANServer As String, iANDigits As Integer

    'Initialize
    strANServer = GetServer
    iANDigits = GetDigits
    Call CheckCounter

    If Not (doc Is Nothing)  Then
        varTemp = doc.GetItemValue (C_AN_NUMBER_FIELD)
        If (varTemp(0) = "") Then
            'This doc has not been assigned a number yet. Attempt to do
            so.
            If SetNumber (doc, strANServer, iANDigits) Then
                'Success
                Call doc.save (False, False)
            Else
                'Failure
                'If desired, log an error at this point
            End If
        Else
            'The doc has already been numbered. Ignore it.
            'Why was this routine called in the first place?
            '  If desired, log a warning message at this point.
        End If
    End If

End Sub 'ANNumberDoc

Private Sub CheckCounter
    'Checks for the counter doc and tries to create it if necessary
    Dim s As New NotesSession, db As NotesDatabase
    Dim docTemp As NotesDocument, vwSettings As NotesView
    Set db = s.CurrentDatabase
    Set vwSettings = db.GetView (C_AN_SET_VIEW)
    Set docTemp = _
        vwSettings.GetDocumentByKey (C_AN_SETKEY_CTR)
    If docTemp Is Nothing Then
        'The counter doc was not found. Create it.
        Set docTemp = db.CreateDocument
        docTemp.Form = C_AN_SET_FORM
        docTemp.fValue = "1" 'Start numbering at 1
        Call docTemp.save (False, False)
    End If
End Sub 'CheckCounter
```

The ANNumberDocs routine is nearly identical to ANNumberDoc, except that it's called by an agent and processes multiple order documents. It expects the agent to pass in a collection of all order documents that are new or modified since the last time the agent ran. From that collection of orders, it only tries to get numbers for the ones whose fDocNumber fields are empty. Since the same documents shouldn't be processed over and over, ANNumberDocs calls the NotesSession.UpdateProcessedDoc method for each document as it looks at it.

Since both ANNumberDoc and ANNumberDocs call SetNumber, it's the only routine that has to interface with the core script library, slANAutoNumbering. This makes it easy to consolidate any error handling code needed by the application. Other than error handling, all this routine does is get a number from AutoNumberGet and convert it to a string using AutoNumberCstr.

```
Private Function SetNumber (doc As NotesDocument, strANServer As
```



```
String, iANDigits As Integer) As Integer
    Dim varTemp As Variant
    Dim iANAttempt As Long, bReturn As Integer

    bReturn = False

    iANAttempt = AutoNumberGet(C_AN_SET_VIEW,
    C_AN_SETKEY_CTR, C_AN_SET_FIELD, strANServer, True)
    If iANAttempt >= 0 Then
        'Success
        bReturn = True
        varTemp = AutoNumberCstr(iANAttempt, iANDigits)
        Call doc.ReplaceItemValue (C_AN_NUMBER_FIELD, varTemp)
    Elseif iANAttempt = C_AN_WRONG_SERVER Then
        'Not really an problem. The number will be assigned later
        Print "Delay: An auto-number could not be assigned yet" _
        + " because this is not the central server."
    Elseif iANAttempt = C_AN_NO_COUNTER Then
        Print "Error: An auto-number could not be assigned" _
        + " because the auto-number counter was not found."
    Elseif iANAttempt = C_AN_NO_ACCESS Then
        'Not really an problem. The number will be assigned later
        Print "Delay: An auto-number could not be assigned yet" _
        + " because you do not have access to update the counter."
    Elseif iANAttempt = C_AN_SAVE_CONFLICT Then
        Print "Error: An auto-number could not be assigned" _
        + " because the counter is unavailable."
    Else
        Print "Error #" + Cstr(iANAttempt) _
        + " occurred. An auto-number could not be assigned."
    End If

    SetNumber = bReturn
End Function 'SetNumber
```

Of course, print statements are only helpful to the Notes client, so you may want to customize SetNumber to respond appropriately to each of the three "clients." For example, it might make sense to add a ClientType parameter, and then handle errors differently in each situation. Notes users could see a MessageBox, Web users could see red HTML, and errors caught during batch processing could be written to a log.

Beyond the AutoNumbering tool

The design of the AutoNumbering tool hinges on an essential principle of replication: when designing a replicated application, you must ensure that any given document will never (or very rarely) be edited on more than one server, since this can cause replication conflicts. This principle applies particularly well to applications that maintain strict sequential numbers, but it applies in other situations as well.

Looking back on the design of the Setting form with replication in mind, I see that so far I've only protected the counter setting from replication conflicts generated by the application itself. I haven't done anything to protect any of the settings from conflicts generated by roaming application administrators. So, one good enhancement might be to designate a central server and have the Setting form's QuerySave event prevent anyone from modifying a setting on a different server. Another solution might be to have the user interface transparently send administrators to the central database whenever they try to open the Settings view. But the consequent performance hit might be unacceptable, depending on the organization's WAN connection speeds between the affected servers.

Applied in a slightly different direction, this principle of replication could be

used to relax the restrictions of the AutoNumbering tool. If your application doesn't require strictly sequential unique numbers across all servers, but you want something a little more readable than Notes/Domino's built-in @Unique function, you could have each server maintain its own count sequence. The body of the SetNumber function could be modified to use the current server's counter setting and append the server's name to each generated number. This would result in numbers like 0002_Paris and 0002_Indianapolis.

A more practical example of split counters might be a feature for tracking Web site usage statistics. It would make sense to count each server's hits separately in order to identify servers that are overloaded or underused.

Clearly, there are many scenarios in which replication can impact a Notes/Domino application's settings and counters. We have not exhausted all the possibilities, but keeping the principles of replication and sound design firmly in mind can go a long way toward keeping any distributed application in one piece.

ABOUT JONATHAN COOMBS

Jonathan is a software developer for **Joseph Graves Associates, Inc.** in Indianapolis. JGA is a full-service consulting firm that delivers quality IT services and customized e-commerce, Internet, and document management software solutions (www.jgraves.com). Jonathan's professional interests include software reuse, Lotus Notes/Domino and Java technologies, and computational linguistics. He can be reached at **jcoombs@jgraves.com**.

