

Mobilizing Domino Data Using REST: Part 1

The first part of a series on mobilizing Domino data for mobile platforms that don't directly support XML-based Web Services (like Android and the iPhone).

Introduction

Back when I worked for Research in Motion, I presented topics at Lotusphere and the View Developer Conference on how to connect to Domino databases from rich client applications on BlackBerry. Research in Motion had some pretty cool tools at the time (MDS Studio) that made it painfully easy – I could build a BlackBerry application that talked to a Domino Web Service in about 3 minutes.

Beginning with BlackBerry Device Software 4.2, Research in Motion added the ability to connect to Web Services from a BlackBerry Java application (using JSR 172), so I added building a BlackBerry Java application to my set of demos. I then wrote a series of articles on this site that documented the whole process. Here are links to the articles:

After I left Research in Motion, I continued to present at the same conferences and added the Midwest Lotus User Group (MWLUG) conference to my standard tour. Because of the Web Services capabilities of Microsoft Visual Studio, it was pretty easy to add a demo of a Windows Mobile application to my presentations as well.

The iPhone and Android platforms were released without the ability to connect to XML-based Web services, so I had to find a different way for those platforms to connect to Domino. Android includes the JSON libraries from json.org, and there were several versions of JSON libraries for the iPhone platform up there as well, so I decided I'd use REST and JSON to accommodate Android and iPhone. At Lotusphere 2010, while recovering from pneumonia, I added an Android demo to the presentation and was even able to show an almost complete (I never finished it) iPhone example as well. At the conference I promised I'd publish an article here demonstrating how I built the Android application, so here it begins (finally). This article is the first part of a new series that's all about how to use REST and JSON to mobilize Domino data on Android. There'll be two articles in this series here, plus I'll eventually publish an article in the View illustrating the iPhone application.

The Sample Application

In case you missed the first part of this series, the application we're building here allows a mobile user to lookup contacts in the Domino Directory. The assumption here is that most every Domino customer has some extra database of contacts and mobile users will need the ability to lookup contact

information. This is the same process as demonstrated in the original series; it's just modified so it leverages JSON instead of Web Services.

About REST

REST stands for Representational State Transfer

(http://en.wikipedia.org/wiki/Representational_State_Transfer) and essentially it's a form of Web Service where parameters for a request are sent on the URL to the server and the server's response is returned in the body of the HTTP response typically in XML or JSON format although it could be in any format. RESTful Web Services are much easier to use than XML Web Services since it's so much less work to call the service and there's much less overhead when the data is returned as JSON rather than XML.

About JSON

JSON stands for JavaScript Object Notation and it's a way of representing data in a textual format that's easy to parse and manipulate. To quote json.org (www.json.org):

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

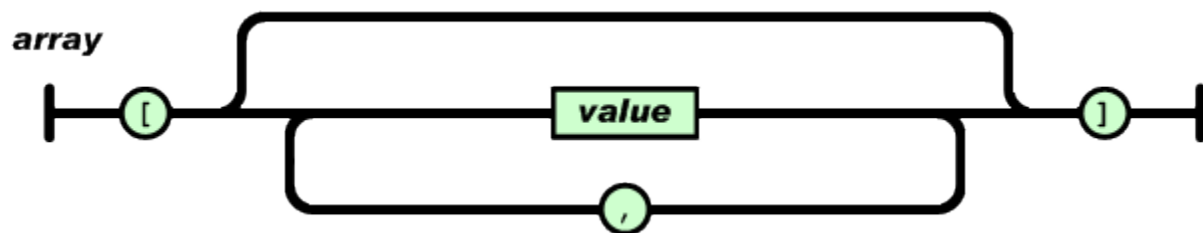
JSON is built on two structures:

- * A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

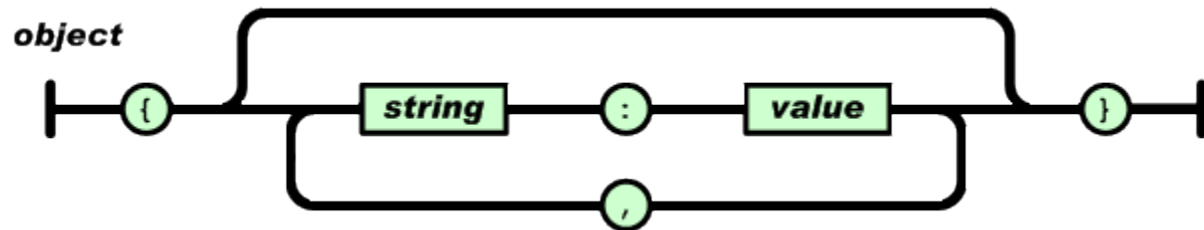
- * An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures."

In JSON, an array looks like this:



And an Object looks like this:



You'll see how these apply in the following section.

*JSON Images shamelessly 'borrowed' from www.json.org.

Designing the JSON Agent

In the Web Services version of the agent I created for the BlackBerry and Windows Mobile versions of the mobile application, I created a single service that supported two operations: GetUserList and GetUserDetails. Since the needs for this application were for there to be the same two operations and each operation would be called by a URL, it would seem to make sense to make a separate Domino web agent for each operation. Unfortunately, that approach would require two agents and a bunch of duplicated code between them, so I decided to create a single agent that exposed the two operations through a single interface. So, there is going to be a single Domino agent called by the mobile application and different results would be returned to the calling program depending on what information is passed on the URL to the Domino server.

So, the URL a mobile application will be using to use the service will look like this:

<http://server/database.nsf/domdirlookuprest?openagent&cmd=COMMAND&searchstr=SEARCHSTRING>

Where the 'cmd=' and 'searchStr=' portions of the URL are places where the calling program can pass in parameters to control what the service returns.

To obtain a list of contacts whose last name begins with 'war', the mobile application will call the following URL:

http://server_name/bbnames.nsf/domdirlookuprest?openagent&cmd=list&searchstr=war

In this case, the cmd is 'list' and the searchStr is 'War' – when the agent runs on the Domino server, it will return the following JSON array (assuming that there are two contacts defined in the database whose last names begin with War):

```
["John Wargo", "Anna Wargo"]
```

Once a mobile application's user selects a contact, the agent is called again with the following URL:

<http://localhost/bbnames.nsf/domdirlookuprest?openagent&cmd=details&searchstr=john+wargo>

In this case, the cmd is 'details' and the searchStr is my contact name. When the agent runs on the Domino server, it will return the following JSON object (notice that the first call, the call to get the list, returns a JSON array, while the second call (for the details) returns an object):

```
{
  "FullName" : "John Wargo",
  "LastName" : "Wargo",
  "FirstName" : "John",
  "EmailAddress" : "jwargo@att.com",
  "OfficePhone" : "330.123.4567",
  "MobilePhone" : "330.987.6543"
}
```

That's it, that's all there is to the agent I'm showing you how to build in this article.

Probably asking yourself "Why not use the JSON capabilities already built into Domino to provide this functionality?" Well, Domino's ability to generate JSON output is designed for rendering views in a format that's easy for an application to process. Using the following URL:

<http://server1/database.nsf.nsf/viewname?ReadViewEntries&OutputFormat=JSON>

would cause the Domino server to render the view as JSON, it doesn't help us here because we're passing in search strings and trying to retrieve only one document. It could be done (I think) with a single category view, but taking that approach was much more work than I wanted to do.

Building the JSON Agent

Let's dig into the agent. It essentially begins with the standard stuff I put into any Domino agent:

```
Option Public
Option Declare
Option Base 1
```

Then I define some common variables I'll need as I process the database view:

```
'Notes Objects
Dim db As NotesDatabase
Dim doc As NotesDocument
Dim s As NotesSession
Dim tmpName As NotesName
Dim userView As NotesView
Dim userDoc As NotesDocument

'Other variables
Dim i As Integer
Dim jsonText As String
```

Finally some constants that are used to prepare the JSON text outputted by the agent:

```

Const amp = |&|
Const BR = |<br />|
Const comma = |, |
Const errorStr = |ERROR: |
Const quoteStr = |"|

```

It's the Initialize subroutine where all of the processing happens. Remember, the service works by having essentially a single URL that's called by the calling program. Parameters passed on the URL instruct the agent what to do. The Initialize subroutine essentially does nothing but parse the URL and call the appropriate subroutine depending on which command ('list' or 'details') is passed to the agent on the URL.

Since we're trying to output JSON in this agent, Initialize begins with a print statement that causes Domino to skip the building of an HTML page to be sent to the calling program. If this wasn't there, Domino would assume the agent was delivering HTML content and would prepend the beginning parts of a web page to the agent's output.

Next, the agent gets a handle to the current Notes Session object, database object then the agent's context (through set doc = s.DocumentContext). After that, the agent gets the URL string from the document context and parses it through repeated calls to the GetCmdLineValue function. Once it knows what command has been sent (list or details) and what search string to use, it calls the appropriate subroutine (GetContactList or GetContactDetails) to lookup the appropriate document(s) and output the necessary JSON text.

```

Sub Initialize()

    'Print out this line to force Domino to not write it's own
    'HTML gunk at the beginning of the resulting page
    Print "Content-Type:text/html"

    Dim cmdName As String
    Dim queryStr As String
    Dim searchStr As String
    Dim tmpStr As string
    Dim tmpInt As integer

    'Initialize our Notes session object
    Set s = New NotesSession
    'Then get a handle to the current database
    Set db = s.CurrentDatabase
    'Get a handle to the agent's context (header variables and so on)
    Set doc = s.DocumentContext

    'Parse the command line and call the correct function
    queryStr = LCase(doc.Query_String_Decoded(0)) & amp
    cmdName = GetCmdLineValue(queryStr, "&cmd=", amp)
    searchStr = GetCmdLineValue(queryStr, "&searchstr=", amp)

```

```

'Launch the appropriate function based upon what's passed in the URL
If cmdName = "details" Then
    GetContactDetails(searchStr)
Else
    If cmdName = "list" Then
        GetContactList(searchStr)
    Else
        'We have an invalid command passed on the Query_String,
        'so return an error
        Print errorStr
    End If
End If

End Sub

```

The GetCmdLineValue function simply takes an input string and returns everything between the two delimiters.

```

Function GetCmdLineValue( textStr As String, delim1 As String, delim2 As
String) As String

    Dim startPos As Integer
    Dim tmpInt As Integer
    Dim valLen As Integer

    'find the first occurrence of the delimiter
    tmpInt = InStr( textStr, delim1)
    'Only continue if we've found something
    If (tmpInt > 0) Then
        'Figure out where the value starts
        startPos = tmpInt + Len(delim1)
        'Then look past there for the second delimiter
        valLen = InStr(startPos, textStr, delim2) - startPos
        'The value we're looking for is between the two delimiters
        GetCmdLineValue = Mid( textStr, startPos, valLen)
    Else
        GetCmdLineValue = ""
    End If
End Function

```

The GetContactList subroutine uses the provided searchString to search the contents of the default Domino Directory \$VIMPeopleByLastName view. If it finds any documents, it builds a JSON array containing each contact name then prints the array; thereby sending the array information to the calling program.

```

sub GetContactList(searchStr As String)

    Const arrayStart = "["

```

```

Const arrayEnd = "]"

'Local Notes objects
Dim ve As NotesViewEntry
Dim vec As NotesViewEntryCollection

'Other variables
Dim numContacts As Integer

'Open the view we're going to lookup against
Set userView = db.GetView("($VIMPeopleByLastName)")
If Not userView Is Nothing Then
  'Do we have a search string?
  If Len(Trim(searchStr)) > 0 Then
    'See if we can find any users by the search string
    Set vec = userView.GetAllEntriesByKey(searchStr)
  Else
    'Otherwise get all documents
    Set vec = userView.AllEntries
  End If
  'Now, if we have any entries - put them into the array
  If vec.Count > 0 Then
    'Get the number of contacts to use throughout the rest of the code
    numContacts = vec.Count
    'Start the Array JSON text
    jsonText = arrayStart
    'Process all of the entries in the View Entry Collection
    For i = 1 To numContacts
      Set ve = vec.GetNthEntry(i)
      If Not ve Is Nothing Then
        Set userDoc = ve.Document
        If Not userDoc Is Nothing Then
          Set tmpName = New NotesName(userDoc.FullName(0))
          If Not tmpName Is Nothing Then
            jsonText = jsonText & quoteStr & tmpName.Common & _
              quoteStr
          Else
            jsonText = jsonText & quoteStr & errorStr & _
              |Unable to obtain Contact Name| & quoteStr
          End If
        Else
          'Unable to get the document
          jsonText = jsonText & errorStr & _
            |Unable to open the contact document| & quoteStr
        End If
      Else
        'Unable to access the View Entry
        jsonText = jsonText & quoteStr & errorStr & _
          |Unable to access the ViewEntry| & quoteStr
      End If
    Next i
  End If
End If

```

```

    End If
    'Add a comma if we need one (when there's more than
    'one name being returned)
    If (i < numContacts) Then
        jsonText = jsonText & comma
    End If
Next i
'Write our resulting JSON results to the browser
Print jsonText & arrayEnd
Else
    'We got nothing, so return an empty array to the
    'calling program
    Print arrayStart & arrayEnd
End If
End If
End sub

```

The GetContactDetails subroutine uses the provided searchString to search the contents of the default Domino Directory \$VIMPeople view. If it finds a document (It should find only one unless there are more than one contacts in the database with the exact same name), it builds a JSON object that contains each of the fields retrieved from the document then prints the object, causing the object to be returned to the calling program. .

```

sub GetContactDetails(searchStr As String)

    Const jsonStart = |{|
    Const jsonEnd = |}|

    'Do we have a search string?
    'the calling program should check, but have to make sure
    If Len(Trim(searchStr)) > 0 Then
        'Open the view we're going to lookup against
        'this is a different view because we're searching against
        'the full name where above we're using last name
        Set userView = db.GetView("($VIMPeople)")
        'Make sure we have the view
        If Not userView Is Nothing Then
            'Try to get the user document using abbreviated full name as a key
            'This should work because the view is sorted that way.
            Set userDoc = userView.GetDocumentByKey(searchStr)
            If Not userDoc Is Nothing Then
                'Start the JSON text we'll be returning
                jsonText = jsonStart
                'Populate the result fields
                Set tmpName = New NotesName(userDoc.FullName(0))
                jsonText = jsonText & |"FullName" : | & quoteStr & _
                tmpName.Abbreviated & quoteStr & comma
                jsonText = jsonText & |"LastName" : | & quoteStr & _
                userDoc.LastName(0) & quoteStr & comma
            End If
        End If
    End If
End sub

```

```
    jsonText = jsonText & |"FirstName" : | & quoteStr & _  
    userDoc.FirstName(0) & quoteStr & comma  
    jsonText = jsonText & |"EmailAddress" : | & quoteStr & _  
    userDoc.InternetAddress(0) & quoteStr & comma  
    jsonText = jsonText & |"OfficePhone" : | & quoteStr & _  
    userDoc.OfficePhoneNumber(0) & quoteStr & comma  
    jsonText = jsonText & |"MobilePhone" : | & quoteStr & _  
    userDoc.CellPhoneNumber(0) & quoteStr & jsonEnd  
    Print jsonText  
Else  
    Print errorStr & |Unable to open Contact Document|  
End If  
Else  
    Print errorStr & |Unable to open User View|  
End If  
Else  
    Print errorStr & |Search String not provided|  
End If  
End sub
```

Conclusion

That's all there is to it. In the next installment, I'll show you how to build an Android application that consumes the service. It only took me a year to deliver on my promise to publish this article. I wonder how long it will take me to publish the next one article. Stay tuned.