The Inside Track
"XML Reprise"
February 2001

The Federal Trade Commission (FTC) has given the acquisition of Great Plains by Microsoft the green light. It's a major milestone. This month Great Plains is hosting another Technical Conference in Fargo. It's a major learning event. The extent of the connection between the two events is that they both occurred in February and that they both are major events that affect the entire Great Plains virtual community.

Learning

The conference affects folks like us a bit more than the non-technical people in the virtual community. Another major impact on us is, or soon will be, XML. At the conference I presented a session called "Getting data to the Web with XML and XSL." In the November Solution Developer Newsletter I wrote about XML and gave a brief example of XSL.

I've been beating my brains for a way to justify reusing the material from my session. I went looking on the Web for the percentage that people retain when they learning something. You know, you only retain something like 40% the first time you hear it and you retain 65% the next time. I didn't find the number, but I didn't look all that hard. I found an interesting article about multiple modes of learning. Ah but what the heck, I just think this stuff is so great and useful.

Well, just a little justification, because I wouldn't want you to think you're getting away with a freebie by not going to the conference (mine was not the best or only session by a long shot) or that having gone to the conference you wasted time sitting through my session (ouch!). This article I found says that in the four learning modes (self, lecture, concurrent and collaborative, see "The Circle of Learning" at http://epaa.asu.edu/epaa/v5n7/) no one mode was superior; they were all complementary. So repeating some information from my session here is a good thing! If you came to the conference, then you'll have two modes, self and lecture, to learn from. Just think how smart you'll be.

Now that I've done a little work on my guilt complex, lets have some fun.

Looking

I wrote this prototype for a lookup in a Web browser. It uses a ton of XML and a pretty fair chunk of XSL. A lookup should allow the user to locate and select information of interest. Often a lookup will provide an abbreviated view of some existing information. Once the user has located the appropriate data, then the whole record is retrieved and displayed.

A lookup needs a fair amount of flexibility since the user may only have partial information about the record being sought. For instance, the user may be trying to pull up a Customer but only knows the customer's phone number. In another instance, perhaps the customer's zip code and date of last order is known. As the size of the database table grows, flexibility becomes more important and performance gets worse.

I had several goals for my prototype. Besides flexibility and performance, I wanted to use XML and XSL and show that XML and XSL could be used successfully with other existing technologies. Applications typically need lots of these lookups, so I wanted to make building them a pretty simple operation, reusing code as much as possible.

Layout

In Figure 1 you will see a simple example of the lookup. This lookup is built with an HTML Table. There are 5 areas of interest, by rows. The top row contains 4 browse buttons. The next row contains a "Find" value and button. Then the column headings followed by the rows of data. The last row contains a "select" button.

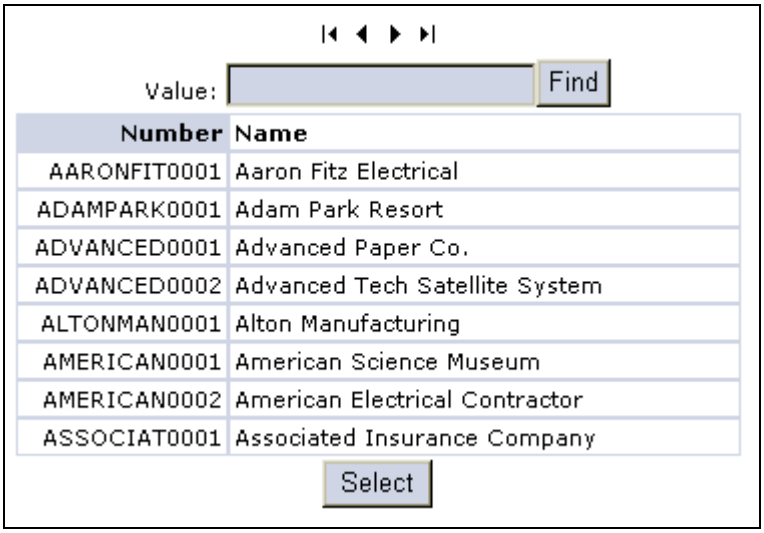


Figure 1

The Next and Prior browse buttons retrieve enough data to fill the lookup with a one record overlap. The overlap allows the lookup to request a block of data in a stateless fashion. With current browser based applications, being stateless on the server is a big plus. If your application is stateless it can easily move from server to server in a server farm and there is less overhead on the server while waiting for the users next interaction.

The find value is used to locate records based on the sorted column. The column headings can be double clicked to change the sort order and by selecting and using drag-and-drop the order of columns can be changed. A data row is selected by double clicking or by single clicking and pressing the select button.

What's really great is that the lookup is defined by XML, the definition is turned into XHTML (HTML that follows the rules of XML) by XSL. Data is requested using XML, the query is defined using XML and the requested data is returned to the lookup as XML. XML everywhere!

Links

All of the files for the prototype are available from the Tech 2001 web site on PartnerSource. The file is called DataWebXMLXSL.zip. You can access all the Tech 2001 materials at:

ftp://ftp.greatplains.com/partner_source/tech2001_winter/

The direct link for my session is:

ftp://ftp.greatplains.com/partner_source/tech2001_winter/DataWebXMLXSL/DataWebXMLXSL.zip

I would like to point out a PowerPoint presentation that should be of interest to the entire Great Plains developer community. On Sunday morning, February 18th, Tim Brookins Great Plains Technical Fellow and Director of Platform Services presented a Tech Conference Super Session titled "Next Generation Technologies". During Tim's session he discussed the motivation, products and technologies that are behind the next generation initiative at Great Plains. A must view session that would be great to see live but extremely valuable nonetheless:

ftp://ftp.greatplains.com/partner_source/tech2001_winter/FutGPTechnlgies/FutGPTechnlgies.ppt

Logic

How does it work? Unzip the Tech Conference .zip file and read the file called README.DOC to install the prototype. Then follow along:

1. A URL is sent to the server. The URL specifies which lookup is requested:
   http://localhost/…/GPLookup.asp?Lookup=Customer

2. GPLookup.asp takes the query string (the stuff after the question mark) and using the value of Lookup, locates the Lookup definition in an XML

file. In the example in the previous step, the XML file will be Customer.xml.

3. GPLookup.asp loads GPLookup.xsl and using the Lookup XML, performs a transform. A transform essentially runs an XSL program. The XSL picks pieces of information out of the XML and uses it in the output of the transform or otherwise modifies the behavior of the program.

   For instance, the Lookup UI XML says how many rows to include in the lookup and the XSL uses that value to generate a certain number of rows in the data portion of the HTML table. The title of the lookup is pulled out of the XML and inserted into the HTML title element.

4. The result of the transform is sent back to the browser where the HTML is rendered. Code in the HTML runs when the browser windows loads to pull in the first chuck of data. Look at Customer.html for an example of the transformed output and at line 31 to see how the initial data is loaded.

5. The vast majority of the functionality of the lookup is held in a separate file called GPLookup.sct as a scriptlet. A scriptlet is like a component written in script (JavaScript in this case). The XSL transform hooks up the scriptlet to the generated HTML so that when the user performs an action, a method in the scriptlet is invoked to respond to the action.

   For instance, if the user clicks on the Top browse button, see line 42 in Customer.html, the Top() method in the scriptlet is called. Many of the methods in the scriptlet retrieve data. All the methods that need data, like Top() on line 236, work by calling another method called Send().

6. The Send() method, starting on line 110, takes information available in its parameters and other information available in the lookup and creates a packet of information formatted as XML.

   One of the important pieces of information that is sent is the name of an XML file that defines the query. The file name came from the Lookup XML and was inserted into the HTML by the XSL, GPLookup.xsl. See line 4 in Customer.xml and line 23 in Customer.html. To keep life simple, I have named the query definition XML file the same as the lookup UI XML file with “Query” stuck in at the end. For instance, for lookup UI XML Customer.xml, the query definition XML file is called CustomerQuery.xml.

   The parameter packet is sent to GPQuery.asp using HTTP. Look at file Params.xml to see an example of the parameter XML.

7. GPQuery.asp receives the packet of XML. It pulls the information out of the XML and creates a COM object called GPQuery.LookupData. It then invokes the GetData() method in the COM object passing along the

parameters it received from Send(). Again, one of the most important parameters is the name of the XML file that contains the query definition. To test GetData() you can run an interactive test program called TestGPQuery found at:

GPQuery\TestGPQuery\bin\debug\TestGPQuery.exe

8. The GetData() method uses the parameters, to build an SQL query that is executed using ADO. The query definition XML file is read and used to determine that ADO connection string, columns and other information needed to build and execute the SQL query. See GPQuery\LookupData.cs to see the source code for the COM component.

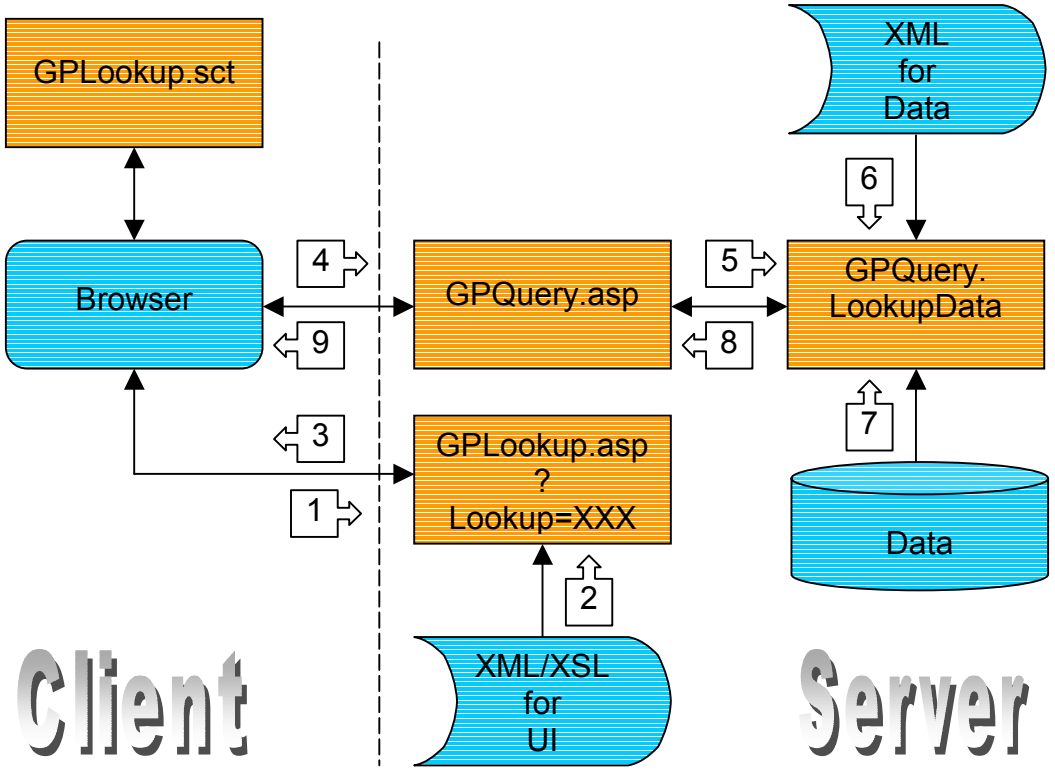
9. The results of the query are turned into an XML string that is the return value of GetData(). See Results.xml as an example.

10. GPQuery.asp gets the return value from GetData() and sends the XML back to the browser using HTTP.

11. The Send() method in the scriptlet on the browser receives the XML and passes it to another method called Populate() starting on line 150 that changes the table data HTML using DHTML.

Linkage

To help bring everything together, here is a picture that includes all these steps.



1) URL invokes GPLookup.asp with the query string.
2) GPLookup.asp transforms requested lookup with GPLookup.xsl.
3) Transform output sent back to the browser for display.
4) User action causes query request to be sent to GPQuery.asp.
5) GPQuery.asp calls GetData() in COM component GPQuery.LookupData.
6) GetData() reads Query Definition file.
7) GetData() executes the SQL query.
8) GetData() return the result as XML to GPQuery.asp
9) GPQuery.asp returns the data to the browser for display in the lookup.

Leaving

From a high level, in a simple prototype like this, I'm almost tempted to say that Internet development won't be so hard. With a streamlined prototype, ignoring a lot of issues, you might be too. But in keeping with the L's that I've used to head each section of this month's column, I want you to remember just one more. Think about all the stuff I "left-out" of this prototype.

Think about pushing data back to the server and making sure it got there securely and completely. Know that Great Plains has teams working on security and transparently supporting transactions.

Think about making this lookup work with a dozen different languages. Know that Great Plains has a team working on metadata and how to represent information in a logical, abstract fashion that allows for simple localization.

Think about customizing the UI per user or per company. Know that Great Plains has a team working with Visual Studio for Application (VSA, which is the .NET equivalent of VBA) and how to expose business functionality safely to customer modification.

Think about integrating your enhancements into a full-blown web enabled application. Know that Great Plains is architecting its next generation products as a cooperating set of component and that who developed the component is not a limitation to what the component can accomplish.

Think about design and management of a large, complex development effort. Know that Great Plains is using high-level design and modeling tools like Rose from Rational so that designers and developers can share information accurately and completely and is creating a documented repeatable development process that can be replicated by partners.

Lots to think about. Lots that the Platform Services folks at Great Plains will build and you will get by using the tools, processes and techniques we will provide.

With so much to think about, it won't be long before I'm back with more on The Inside Track.

Later,
Karl Gunderson
Technical Evangelist