

SAS Intrnet © to XML to Visual Basic ©

By Anthony Hughes – Abacus Technical Services, Inc.

Abstract

This paper will describe model architecture in use at Texas Instruments Corp. to distribute information from the factory floor to an end-user's Personal Computer.

The model architecture uses several components, including SAS Institute products, Microsoft Corp. tools such as Visual Basic©, and web-oriented Information Technology languages such as HTML and XML.

Data that describes factory-floor operations at TI is stored on a Unix-based Sun© computer. The data is managed by components of the SAS© product family, which provides remote access to the data through the SAS/Intrnet© and SAS/SHARE*NET© products, among others.

A Web server is constantly running. It receives an incoming Web request from a client application and then sends a request for service to a continuously running SAS session, called the Application Server, a component of SAS/Intrnet. (SAS/SHARE*NET is also needed, since the web server is on a different computer than the SAS system and its data.)

The Application Server then executes a SAS program to satisfy the request. The Application Server retrieves the name of the program, and any other necessary parameters, from the text of the web request, places a SAS Macro variable into the session corresponding to each parameter it received from the web request, and invokes the requested program. The code of the SAS program is responsible for reading the Macro variables, processing the request, creating output in the form of XML, and returning the XML to the web server and, ultimately, the client application.

The client application is actually a Visual Basic program running on a user's Personal Computer. It presents a graphic interface to the user that is used to view and control the data, issues web-based requests to the web server, and interprets the XML code that the web server returns.

Introduction

Here at Texas Instruments, there are many SAS programs that are invoked via a web-based request through the company's internal intranet. Some of them are complex, some are lengthy and have many features, but my example will feature a program that maintains a simple table stored in a SAS Data Set.

I will describe these things:

- The program and its behaviors
- The coding techniques needed to issue a web request from a Visual Basic program
- How data can be structured into a format known as XML
- How a Visual Basic program can work with (parse) XML data
- How to code a Base SAS program to generate XML.

Definitions

Application Server

This is a SAS session that is running continuously and waiting for a request from the Broker for a SAS service. When it receives a request, it will execute a SAS program specified by the request. The Broker will supply any parameters from the web request to the Application Server. The Application Server will then execute the SAS program requested by the Broker, and place each parameter passed to it by the Broker into a similarly-named SAS Macro variable.

Broker

This is a program that is invoked by the web server from an incoming web request.

Many times, a web request is for the text of a specific web page that the web server has in its files, but sometimes the request is for the execution of a specific program.

The process of executing a program to satisfy a web request is called CGI.

If the web page specifies the Broker as the CGI program to be executed, then a SAS session will ultimately take place. The broker will notify the Application Server that a SAS program must be executed, pass the name of the program to the Application Server, and pass it any parameters that came with the web request.

The broker can do much more. It can be configured to choose among several Application Servers running simultaneously, depending upon workload, and it can even be set to launch a new instance of an Application Server when required.

CGI

The technology that causes a web server to execute a program instead of merely passing back a prewritten web page is called CGI.

A web server is a program that is running continually, and is waiting for a message from an external network. The message will be assigned a code number, called a PORT, which identifies the message as a web request. Very often, port 80 indicates that the message is a web request.

Common examples of web servers are Apache, Microsoft IIS (Internet Information Server), Xitami, and others.

The web server knows whether to treat a particular message as a request to execute a program because a special file, a configuration file, will have defined the paths containing executable programs. If the web request refers to one of these paths, the server will attempt to execute it as CGI.

Control

A control is an element of a Graphic User Interface, i.e., a screen. It can be a text box that a user can type characters into, a list with selectable lines, a picture, etc.

Dictionary

This is a data structure available to a Visual Basic program. It is possible to use a dictionary to create a list of things, give them names, and manipulate the list in some interesting ways. The elements of the list can be anything, even other dictionaries.

A dictionary is very useful because XML code can be HIERARCHICAL in nature, and dictionary type data structures can contain a representation of a full XML file.

DOM (Document Object Model)

The popular web browsers have an underlying structure called an object model, or specifically, the Document Object Model.

A web page that contains a JavaScript program can manipulate pieces of the model and create special effects on the browser.

A portion of this structure is used to support XML in particular. An application program written in Visual Basic that contains an embedded web browser can read this underlying structure and extract the data from the XML code.

In the case of XML, the underlying structure is called XMLDOM.

Fileref

The abbreviation for a path in a computer's file system. For example, the fileref "MySrc" is an abbreviation for a path such as '/sas/sasv82/source'. Filerefs are assigned in a SAS session by use of the FILENAME verb.

Some common filerefs as are used in this paper are PROGRAMS, for the location of Base SAS source code that will be executed by the Application Server, and _webout, the location to which a SAS program will write web pages and XML code.

Form (HTML)

A web page that allows you to press a button and return information, is called a Form.

For example, you enter your e-mail address, a password, and some other information onto a web page, press a button, and the browser returns your current bank balance.

Strictly speaking, the web page is said to **CONTAIN** a form.

The form consists of special HTML codes that describe the data fields you are returning, and describe how the data is to be sent.

The Visual Basic programs that communicate with the Unix computers in our model architecture do so by simulating form-based processing. That is, they send certain parameters to a web server and request that something be done with it.

Form (VB)

The visual screen you see in a Visual Basic program is usually referred to as a **FORM**. It has no relationship to the term **FORM** as used in web requests. Just a coincidence.

HTML

This is a set of codes that describe the appearance and functionality of a web page. The HTML code is interpreted by a web browser on a user's PC.

HTTP

This is a set of codes that describe what sort of data a web transmission contains and how it is to be transmitted and interpreted. It also refers to the set of common processes that underlie the workings of the web itself.

MSXML

This is the file that must be present in order for a Visual Basic program to interpret XML.

Object

A small program that has been embedded into another program. The internal code of the small program is normally not visible to the user of a larger program.

The advantages of using an object rather than just inserting more code into a program are many, including the ability to create many copies of the same object, each with a slightly different identity, into the same program.

SAS/Intrnet

A set of products from the SAS Institute that make it possible for a web server to communicate with a running SAS session.

SAS/SHARE*NET

A component of SAS/Intrnet. If your web server and your SAS system are on different computers, you need SAS/SHARE*NET.

TCP/IP

The common term for the organization and structure of the methods used for the entire internet, not just the web. It stands for T(ransmission) C(ontrol) P(rotocol)/ I(nternet) P(rotocol). A protocol is a way of organizing data and behavior over transmission lines.

Visual Basic

A commonly used language and development environment for personal computers. The term "Visual" refers to its strong emphasis on creating user interfaces. It is proprietary to the Microsoft Corporation. It is less flexible than lower-level languages such as C++, but much faster to use, and thus much less expensive.

Web Server

A program running continuously on a computer, listening for incoming requests from some other computer, for web pages. The pages may already have been written, and are merely being stored, or must be created at the point in time that they are requested, by running a program called a CGI program. The web server will know whether it needs to return an existing web page or must execute a CGI program to create a new web page.

XML

A set of codes, usually with a well-defined syntax, used to specify data and its hierarchical relationships. It is similar to HTML, but is used to transport data between web (and other) applications. Note that XML is for data, but HTML, which looks like XML, is used for web page formatting and markup.

XMLDOM

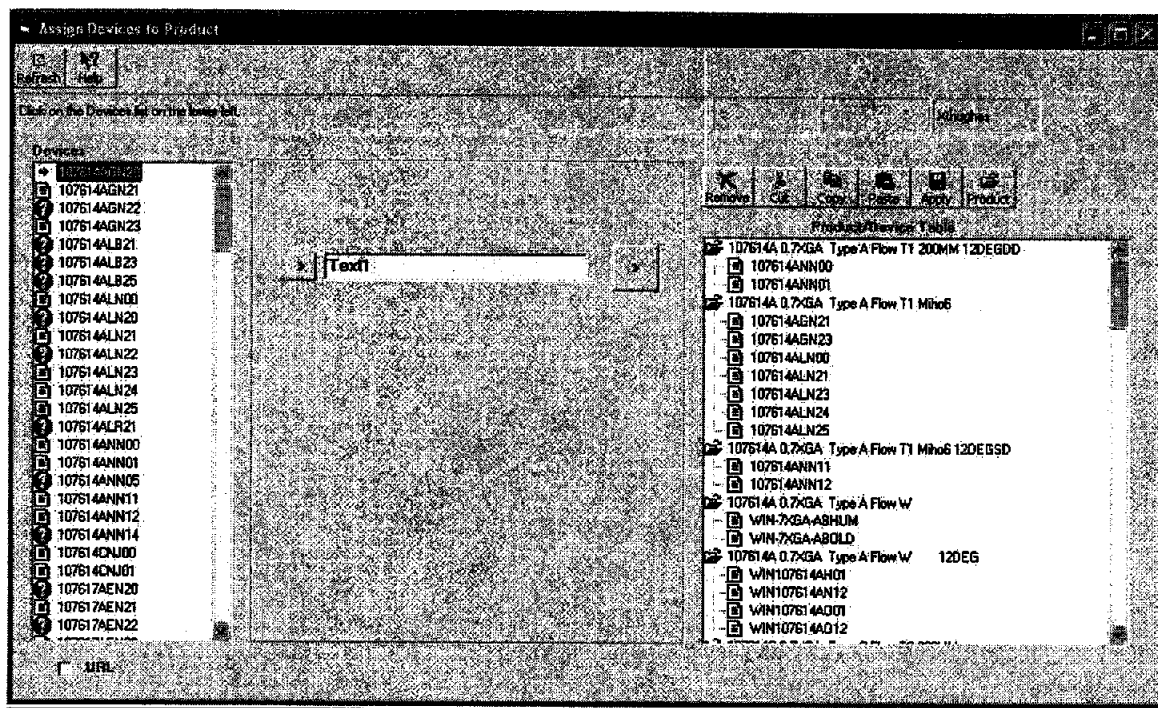
This is the set of programming functions and the data elements that a Visual Basic (and other) program will use to access XML data. It stands for XML D(ocument) O(bject) M(odel).

Architecture

Visual Basic Program Description

There are many programs and web pages in use here at TI that invoke SAS on Unix to return web content. In order to provide a short and concise description of the overall architecture under which these products operate, I will describe one such program, the Define Product application.

Figure 1 - Define Product VB program



This is a picture of the graphic user interface (screen) of the Define Product Visual Basic program.

The purpose of this program is to maintain a table of Product names and their associated device names. A "device" is a specific kind of chip, and a "product" is a designation for a group of devices that are similar to one another. The program will

- display the current table,
- display a list of available devices,
- allow a user to add a device to, or remove it from, a product,
- add a product to the list, and
- remove a product from the list

The table is stored in a SAS Data Set on a Unix-based Sun computer.

A list of available devices is currently held in another SAS Data Set, called "device_list".

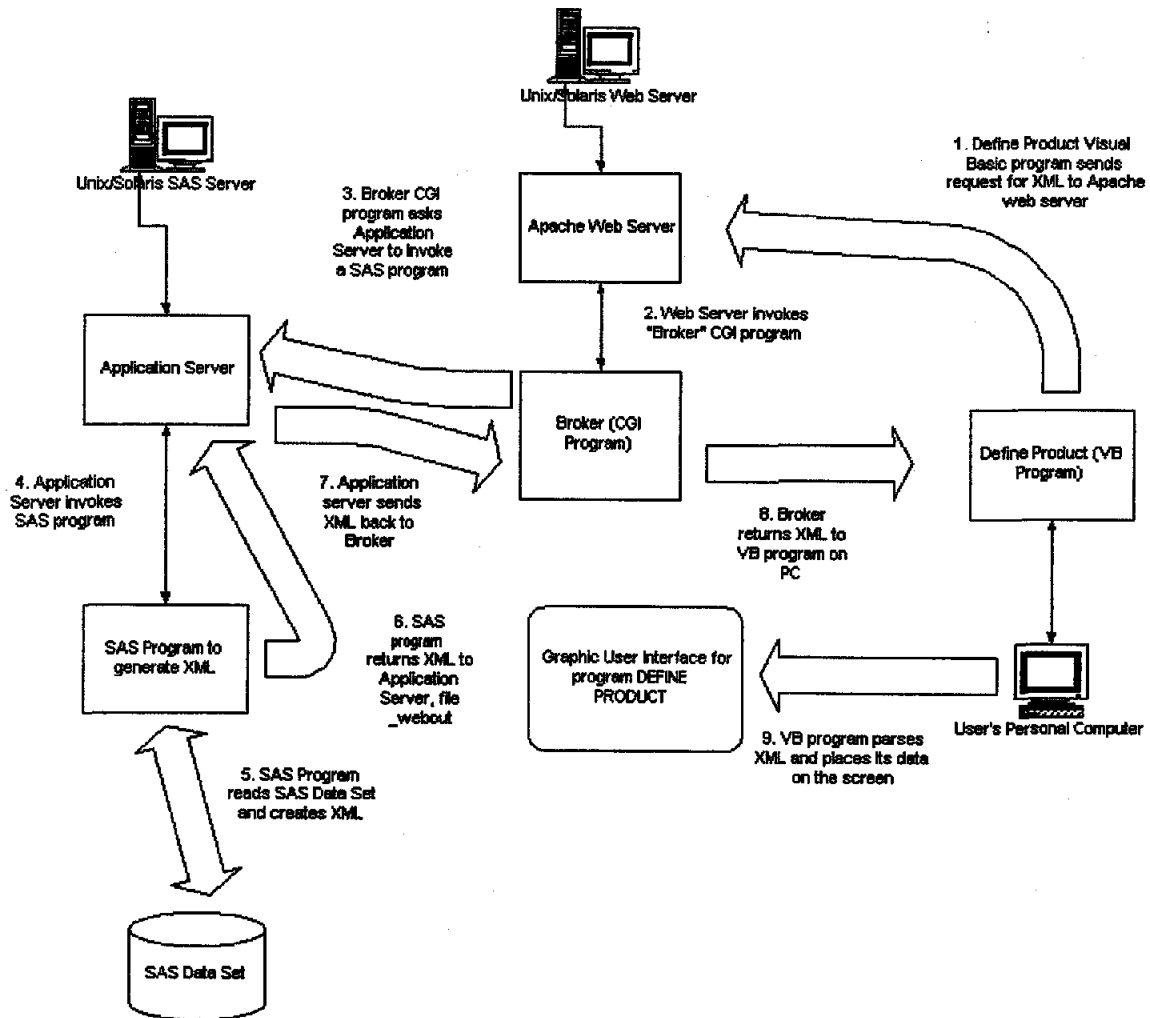
When the program first starts, it issues a web request for a Device list, and the web server returns XML code representing the list of available devices. The program reads the XML, parses it and builds the device list on the left side of the screen.

Immediately afterwards, the program issues a web-based request for a product list. The web server returns XML code for all products, and assigned devices, in the current table and the Visual Basic program constructs the list on the right from the XML.

Other behaviors are initiated by the program user,

The user will click on an item in the list on the left, placing it into the text box on the center. When the user clicks on a Product (top-level item in the tree structure on the right), and then clicks on the large button to the right of the text box, the item in the text box is placed under the Product in the Tree.

XML Request Architecture



The graphic above will illustrate the basic flows of data and control that allow a Visual Basic program to retrieve SAS data from a SAS server.

1. Define Product Visual Basic program sends request for XML to Apache web server.

The Apache web server is continuously running, and is listening for a request for web pages on the TCP/IP connection between itself and the user's PC. The Visual Basic program will not, however, ask for just a web page. It will ask the Apache web server to execute a program, called Broker, that will eventually return XML data to the Visual Basic program.

The request will carry several parameters within itself, such as the name of the requested SAS program to be executed by the SAS Application Server, the name of the program that the web server itself will execute (in this case, "Broker"), any additional parameters to be passed to the requested SAS program, and some additional parameters to be passed to the Application Server itself.

2. Web Server invokes Broker CGI program.

As I have mentioned before, the web server will not try to find a specific web page and return it to the Visual Basic program. Instead, it will execute a program called Broker, which has been placed into the web server's file space. Broker will send a request to execute a SAS program to the Application Server, a program always running, and running on a different computer than the Broker.

3. Broker CGI program asks Application Server to invoke a specific SAS program.

Broker is supplied by SAS Institute for the sole purpose of passing on a request for processing to the SAS system. Rather than have Broker invoke a SAS session every time it is called, Broker will communicate with a continuously running SAS program called the Application Server, through a TCP/IP connection. This has several advantages, such as providing faster response, and allowing Broker to select a particular Application Server session, and even to start a new Application Server session.

4. Application Server invokes SAS program

The Application Server is actually just a SAS procedure called Proc AppServ, that knows about several things. It knows where its SAS programs are stored, since it has defined a fileref called "programs". Note that the source code of all the programs that the Application Server will execute must be present in this fileref.

The Application Server is not restricted to execution of Base SAS code, it can also execute programs written in SCL. The advantages of SCL are: 1. It is faster, 2. It is more secure, since a user cannot modify it, 3. It is more powerful, with stronger coding features, including Object Oriented constructs.

5. SAS Program reads SAS Data Set and creates XML

In this paper, the program we are describing is simple. Its task is merely to add, change and remove entries in a single table (SAS Data Set) that assigns a name (Product) to a group of device codes. When it completes a task, the program will create some XML code and return it to the calling VB program to indicate that the task completed normally. Very often, the XML code will contain a hierarchical description of the contents of the SAS data set.

6. The program will write the XML code to a fileref called `_WEBOUT`. Any XML or HTML code that is written to `_WEBOUT` will ultimately be sent to the Visual Basic program as a response to its request.

This fileref will be assigned by the Application Server. It must not be assigned by the application SAS program.

7. Application Server sends contents of `_WEBOUT` back to BrokerIt is a convention in web-based processing that a web browser running on a user's PC should only communicate with the computer that first spoke to the browser. Speak only when, and to whom, spoken to. So, when you fill out credit card info on a web page and press a button to submit it, the information only goes back to the computer that originally sent the web page. For this reason, the XML that our SAS program has generated is sent back to the Broker, who is patiently waiting for it. The Broker will then pass on the XML to the Visual Basic program that originally requested it. The Application Server will never communicate directly with the Visual Basic program, only the Broker communicates with the VB program.

8. Broker returns XML to VB program on PC

The broker sends back the XML and the entire XML set is placed into the program's memory. The VB program will have been waiting for the Broker to return the data, and the VB program's screen will be locked until the XML returns. This is called "synchronous" processing.

Since the entire XML set will be in the program's memory, there is a real limit for the amount of data the program can easily work with. I have found that about 10,000 rows is a reasonable amount to put into a VB program's memory.

9. VB program parses XML and places its data on the screen

The VB program must now read the data carried in the XML and place it into the appropriate screen controls.

The functions and subroutines that the Visual Basic program must use to extract the data from the XML are supplied by the MSXML file. The overall architectural plan of MSXML is called the XMLDOM. A description of how to use the XMLDOM is available from the Microsoft website. For some practical examples of the use of MSXML, consult the coding description later in this document.

Figure 2 - Sample XML for Device list

```
<?xml version="1.0" ?>
  <document date-created="24MAY02" time-created="13:19:42">
    <libname name="params">
      <memname name="device list" nobs="      164">
        <device value="107614AGN20"/>
        <device value="107614AGN21"/>
        <device value="107614AGN22"/>
        <device value="107614AGN23"/>
        <device value="107614ALB21"/>
        <device value="107614ALB23"/>
        <device value="107614ALB25"/>
        <device value="107614ALN00"/>
        <device value="107614ALN20"/>
        <device value="107614ALN21"/>
        <device value="107614ALN22"/>
        <device value="107614ALN23"/>
        <device value="X1210AAIN8"/>
        <device value="X1272ANE06"/>
        <device value="X1272ANNR6"/>
        <device value="X1272CNNO6"/>
        <device value="X8460AACJ6"/>
        <device value="X8460AACJ6MIHO"/>
        <device value="X8460CACJ6"/>
      </memname>
    </libname>
  </document>
```

Figure 3 - Sample Product XML

```
<libname name="params">

  <memname name="product_list" nobs=" 77"

  >

    <product value="107614A 0.7XGA Type A Flow T1 200MM 12DEGDD" productc="001" userid="xthughes">

      <device value="107614ANN00"/>

      <device value="107614ANN01"/>

    </product>

    <product value="107614A 0.7XGA Type A Flow T1 Miho6" productc="002" userid="xthughes">

      <device value="107614AGN21"/>

      <device value="107614AGN23"/>

      <device value="107614ALN00"/>

      <device value="107614ALN21"/>

      <device value="107614ALN23"/>

      <device value="107614ALN24"/>

      <device value="107614ALN25"/>

    </product>

    <product value="107614A 0.7XGA Type A Flow T1 Miho6 12DEGSD" productc="003" userid="xthughes">

      <device value="107614ANN11"/>

      <device value="107614ANN12"/>

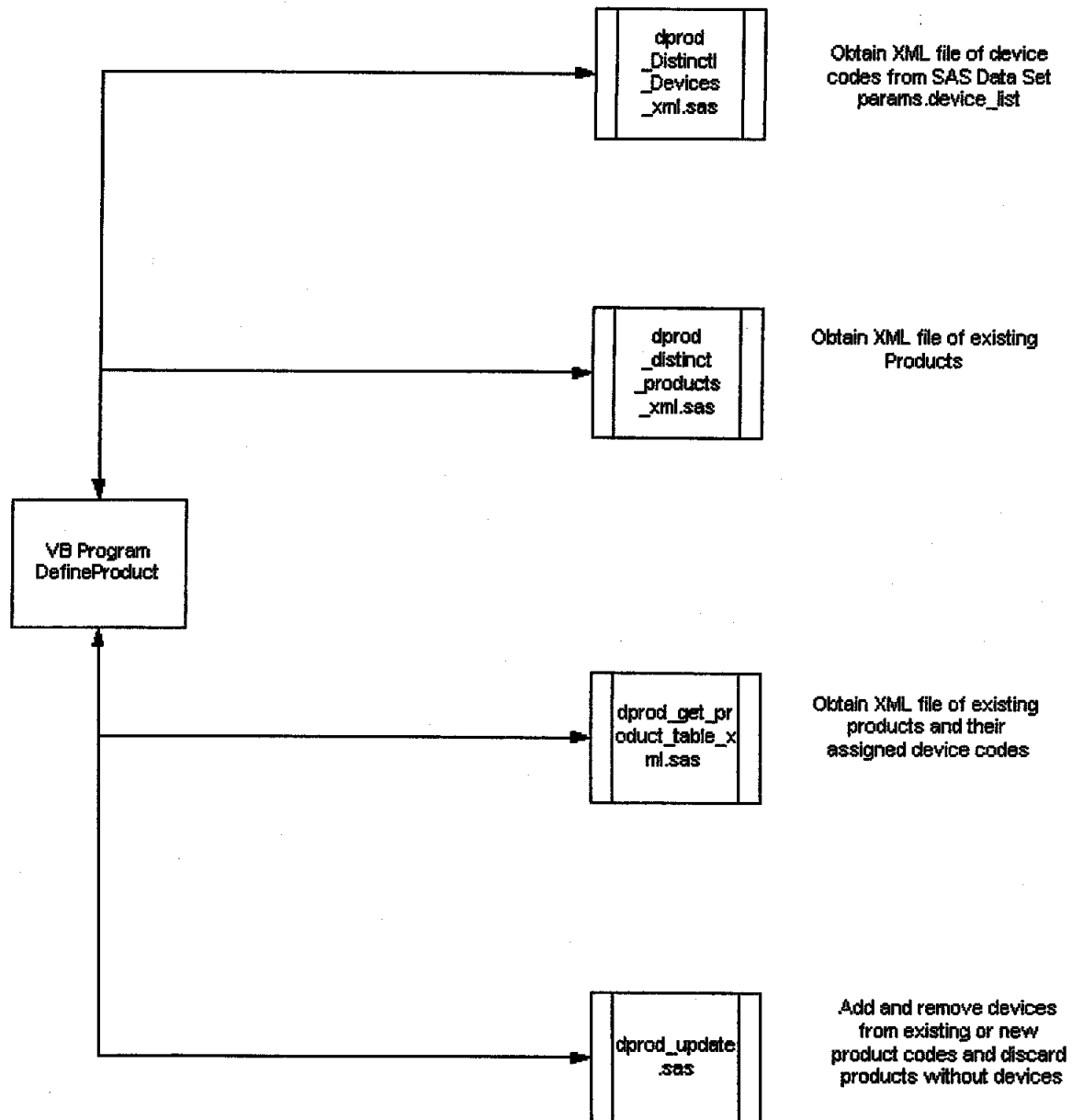
    </product>

  </memname>

</libname>

</document>
```

SAS Modules invoked by VB Program



This figure shows the SAS programs that the VB program DefineProduct invokes. The invocation occurs when the VB program issues a request for XML from the web server. The web server invokes the Broker, and the Broker forces a program execution by the Application Server on the SAS Unix computer.

Figure 4 - VB code to load Device table from XML

```
001 Sub Refresh_DeviceList()
002 'On Error Resume Next

003 Dim nodX As Node
004 Dim Libname, Memname As Object
005 Dim xmldoc As Object
006 Dim myErr As Object

007 Dim Root As Object

008 Dim field As Object
009 Dim value As Object

010 Dim DeviceName, UserId, nobs As String
011 Dim strCommand As String

012 tvwDevices.Nodes.Clear

013 Set xmldoc = CreateObject("Microsoft.XMLDOM")
014 xmldoc.async = False
015 xmldoc.validateOnParse = True

016 URL = "http://dmdweb.de.sc.ti.com/cgi-bin/broker?_service=beta&_program=programs.dprod_distinct_devices_xml.sas&_debug=000" + "
&userid=" + gloUserId + "&ver=" + gloVersion

017 xmldoc.Load (URL)

018 Set myErr = xmldoc.parseError
019 If (myErr.errorCode <> 0) Then

020     MsgBox ("Program: programs.dprod_distinct_devices_xml.sas failed to execute on Unix")
021     Exit Sub
022 End If

023 'Set root to the XML document's root element, COLLECTION:

024 Set Root = xmldoc.documentElement

025 ' place the xml into the web browser

026 Set Libname = Root.firstChild
027 Set Memname = Libname.firstChild

028 nobs = Memname.getAttribute("nobs")

029 If nobs = "0" Then
030     MsgBox "Cannot get a list of distinct values for wafermaps because the table has no records."

031     Exit Sub
032 End If

033 Set field = Memname.firstChild

034 For Each value In Memname.childNodes
035     DeviceName = value.getAttribute("value")
036     Set nodX = tvwDevices.Nodes.Add(, DeviceName, DeviceName, "leafw", "rightarrow")

037 Next
038 End Sub
```

The VB code above will request XML from SAS on Unix that contains a list of devices.

Figure 5 - Create XML for Product/Device list

```

/*=====
Name:      dprod_get_product_table_xml.sas
Purpose:   Download the product/device hierarchical table
Maintenance: Feb 27 2002 - Original code
*/

%include '/sas/sasv82/autoexec.sas'; /* assign the standard libraries */
libname params '/sas/DMD/int_yield/params';
options mlogic mprint symbolgen linesize=120;

%let mynobs=0; /* default value */

run;      /* set the mynobs macro variable */

proc sort data=params.proddefs out=a; by product;
run;
data _null_;
    set a nobs=mynobs;
    call symput('mynobs',put(mynobs,9.));
run;
data _null_;
    length string memtype memlabel $200;
    dquote="";
    file _webout;

    put @1 '<?xml version="1.0" ?>';

    string='<document date-created='
        ||trim(left(put(today(),date7.)))
        ||'" time-created='
        ||trim(left(put(time(),time.)))
        ||"'>';

    put @5 string;
    string = '<libname name='||dquote||"params"||dquote||">';
    put @5 string;

    string = '<memname name='||dquote||"product_list"||dquote
        ||' nobs='||dquote||"&mynobs."||dquote
        ||'>';
    put @13 string;

    put @13 '>';

data _null_;
    length string $200;
    dquote="";
    set a;
    by product;
    file _webout;

    if first.product
    then
    do;
        string=trim(left(product));
        if string='&nbsp;' then string='';
        string="<product"
            ||" value="||dquote||trim(left(string))||dquote
            ||" product="||dquote||trim(left(product))||dquote
            ||" userid="||dquote||trim(left(userid))||dquote
            ||">";

        put @17 string;
    end;

    string="<device value="||dquote||trim(left(device))||dquote||"/>";

```

```

        put @22 string;

        if last.product then put @17 "</product>";
data _null_;
    file _webout;

    put @9 "</memname>"; put @5 "</libname>"; put @5 "</document>";
run;
%include '/sas/sasv82/intrnet/programs/beta/log_userid.sas';
run;

```

Obtaining the user's Logon ID

The users logon ID must be passed to every web-based request for this product. This information is useful as a way of identifying the users of a product, and to determine which features of the product are most useful to them. Also, it may be important to determine how many people use the product, and how often.

Figure 6 - Windows GetUserName function declaration

```

Private Declare Function GetUserName Lib "advapi32.dll" _
    Alias "GetUserNameA" _
    (ByVal lpBuffer As String, nSize As Long) As Long

```

The code above establishes an interface to a function in the Windows operating system that provides the user's logon ID. Note that the function must be declared as 'Private' in order for your Visual Basic program to compile.

Figure 7 - VB code to get user name from Windows

```

Public Function UserName() As String
    On Error GoTo ErrorRoutineErr

    Dim slUserName As String
    Dim st As String
    Dim slCnt As Long
    Dim slDI As Long
    Dim slPos As Single

    '----- get the user's name-----

    slCnt = 199
    st = String(200, 0)
    slDI = GetUserName(st, slCnt)
    slUserName = Left(st, slCnt) & slCnt
    slPos = InStr(1, slUserName, Chr(0))
    If slPos > 0 Then
        UserName = Left(slUserName, slPos - 1)
    End If
ErrorRoutineResume:
    Exit Function
ErrorRoutineErr:
    MsgBox "UserName Error: " + Err + Error
End Function

```

Figure 8 – Generate XML describing devices

```
/*-----  
Name:      dprod_distinct_devices_xml.sas  
Purpose:    Download a list of devices from params.device_list  
Maintenance: Feb 26 2002 - Original code  
-----*/  
  
%include '/sas/sasv82/autoexec.sas'; /* assign the standard libraries */  
  
libname params '/sas/DMD/int_yield/params';  
  
options mlogic mprint symbolgen linesize=120;  
  
%let mynobs=0; /* default value */  
  
run; /* et the mynobs macro variable */  
  
proc sort data=params.device_list(keep=device) nodupkey out=a; by device;  
  
data _null_;  
  
    set a nobs=mynobs;  
  
    call symput('mynobs',put(mynobs,9.));  
  
run;  
  
data _null_;  
  
    length string memtype memlabel $200;  
  
    dquote='\"';  
  
    file _webout;  
  
  
    put @1 '<?xml version="1.0" ?>';  
  
  
    string='<document date-created="'  
        ||trim(left(put(today(),date7.)))  
        ||'" time-created="'  
        ||trim(left(put(time(),time.)))  
        ||'">';  
  
    put @5 string;  
  
    string = '<libname name='||dquote||"params"||dquote||'>';
```

```

put @5 string;

string = '<memname name="||dquote||"device_list"||dquote
      ||' nobs="||dquote||"&mynobs."||dquote
      ;

put @13 string;

put @13    >';

data _null_;

length string memtype memlabel $200;

dquote="";

set a;

file _webout;

string=trim(left(device));

*if string le " " then string='-';

if string='&nbsp;' then string=' ';

string="<device value="||dquote||trim(left(string))||dquote||"/>";

put @17 string;

data _null_;

file _webout;

put @9 "</memname>";

put @5 "</libname>";

put @5 "</document>";

run;

```

Figure 9 - SAS code to create Product XML

```
/*-----  
Name:      dprod_get_product_table_xml.sas  
Purpose:    Download the product/device hierarchical table  
Maintenance: Feb 27 2002 - Original code  
-----*/  
  
%include '/sas/sasv82/autoexec.sas'; /* assign the standard libraries */  
  
libname params '/sas/DMD/int_yield/params';  
options mlogic mprint symbolgen linesize=120;  
%let mynobs=0; /* default value */  
  
run; /* set the mynobs macro variable */  
  
proc sort data=params.proddefs out=a; by product;  
run;  
  
data _null_;  
    set a nobs=mynobs;  
    call symput('mynobs',put(mynobs,9.));  
run;  
  
%macro doit;  
data _null_;  
    length string memtype memlabel $200;  
    dquote='";  
    file _webout;  
  
    put @1 '<?xml version="1.0" ?>';  
  
    string='<document date-created="'  
        ||trim(left(put(today(),date7.)))
```

```

        ||" time-created="

        ||trim(left(put(time()),time)))

        ||">';

put @5 string;

string = '<libname name=||dquote||"params"||dquote||">';

put @5 string;

string = '<memname name=||dquote||"product_list"||dquote

        ||' nobs=||dquote||"&mynobs."||dquote

        ;

put @13 string;

put @13    '>';

data _null_;

    length string $200;

    dquote='";

    set a;

        by product;

    file _webout;

    if first.product

    then

    do;

        string=trim(left(product));

        if string='&nbsp;' then string=' ';

        string="<product"

            ||" value="||dquote||trim(left(string))||dquote

            ||" productc="||dquote||trim(left(productc))||dquote

            ||" userid="||dquote||trim(left(userid))||dquote

```

```

        ||">";

    put @17 string;

end;

string="<device value="||dquote||trim(left(device))||dquote||"/>";

put @22 string;

if last.product then put @17 "</product>";

data _null_;

    file _webout;

    put @9 "</memname>";

    put @5 "</libname>";

    put @5 "</document>";

run;

```

Figure 10 – VB – Fetch Product XML from Unix

```
Sub Refresh_ProductTable()
'Debug.Print "Now entering Refresh_ProductList  "
'On Error Resume Next

Dim Libname, Memname, Product, Device, objX As Object
Dim nodX, nodY, nodZ As Node
Dim xmldoc As Object
Dim myErr As Object

Dim Root As Object

Dim field As Object
Dim value As Object

Dim ProductName, Productc, UserId, nobs, DeviceName As String
Dim strCommand As String

Set xmldoc = CreateObject("Microsoft.XMLDOM")
xmldoc.async = False
xmldoc.validateOnParse = True

URL = "http://dmdweb.de.sc.ti.com/cgi-bin/broker?_service=beta&_program=programs.dprod_get_product_table_xml.sas&_debug=000" + "
&userid=" + gloUserId + "&ver=" + gloVersion

xmldoc.Load (URL)

Set myErr = xmldoc.parseError
'Debug.Print "myErr.errorCode " + CStr(myErr.errorCode)
If (myErr.errorCode <> 0) Then

    MsgBox ("Program: programs.dprod_get_product_table_xml.sas failed to execute on Unix")

    Exit Sub
End If

'Set root to the XML document's root element, COLLECTION:

Set Root = xmldoc.documentElement

' place the xml into the web browser
' MsgBox (Root.xml)

Set Libname = Root.firstChild
Set Memname = Libname.firstChild

nobs = Memname.getAttribute("nobs")
'Debug.Print "Nobs: " + CStr(nobs)
If nobs = "0" Then
    MsgBox "Cannot get rows for the Product Table because the table has no records."

    Exit Sub
End If

tvwProductTable.Nodes.Clear

For Each Product In Memname.childNodes
```

```

    ProductName = Product.getattribute("value")
    Productc = Product.getattribute("productc")
    UserId = Product.getattribute("userid")
    Set nodX = tvwProductTable.Nodes.Add(, , ProductName, ProductName, "openfolderg", "rightarrow")
    nodX.Tag = "product|" + UserId + "|" + Productc

    nodX.Expanded = True

    For Each Device In Product.childnodes
        DeviceName = Device.getattribute("value")
        Set nodY = tvwProductTable.Nodes.Add(ProductName, tvwChild, ProductName + "|" + DeviceName, DeviceName, "leafw", "rightarrow")
        nodY.Tag = "device"

    Next

Next

End Sub

```

Figure 11 - VB code – Update Product

```

Sub UpdateProduct(strProductName)
    ' On Error Resume Next

    Debug.Print "Now entering UpdateProduct"

    Dim nodX, nodY As Node
    Dim XmlHTTP As Object
    Dim xmldoc As Object
    Dim myErr As Object

    Dim nobS As String
    Dim strSql, strString As String

    Dim strName As String
    Dim strValue As String
    Dim strFields As String
    Dim strCsvLabel As String
    Dim strUrl As String
    Dim strFormData As String
    Dim vFormData As Variant
    Dim varUrl, varFormData, varFormData2 As Variant

    Dim strTablename As String

    Dim intX, intY, intYCorrected As Long

    Dim a As Integer
    Dim strDevices As String

    strDevices = ""

    For Each nodY In tvwProductTable.Nodes
        If Mid(nodY.Tag, 1, 6) = "device" Then
            Set nodX = nodY.Parent
            If nodX.Key = strProductName Then strDevices = nodY.Key + "|"
        End If
    Next

    strProductName = CStr(Replace(strProductName, "&", "+")) ' change any ampersands to plus signs

    strUrl = "http://dmdweb.de.sc.ti.com/cgi-bin/broker"
    vFormData = "_service=beta&_program=programs.dprod_update.sas&_debug=000&userid=" + gloUserId + "&ProductName=" + strProductName +
    "&devices=" + strDevices + "&ver=" + gloVersion

```

```

URL.Text = strUrl + "?" + vFormData

' #####
' This is the xmldom version of the request
' #####

' Set xmldoc = CreateObject("Microsoft.XMLDOM")
' xmldoc.async = False
' xmldoc.validateOnParse = False

' xmldoc.Load (Url)

' #####
' this is the xmlhttp version of the request
' #####

Set XmlHTTP = CreateObject("Microsoft.XMLHTTP")
XmlHTTP.Open "POST", strUrl, False
XmlHTTP.send vFormData
Set xmldoc = XmlHTTP.responseXML

Set myErr = xmldoc.parseError
If (myErr.errorCode <> 0) Then
    MsgBox ("Program: wmap_update.sas failed to execute on Unix")
    MsgBox ("source: " + myErr.srctext)
    StatusBar1.Panels(1) = "Product " + strProductName + " failed to be added/updated to product table on Unix."
    Exit Sub
End If

StatusBar1.Panels(1) = "Product " + strProductName + " added/updated to product table on Unix."

End Sub

```

Figure 12 - VB code - delete a Product

```

Sub Delete_Product(strProductName)
    On Error Resume Next

    Dim Libname, Memname As Object
    Dim xmldoc As Object
    Dim myErr As Object

    Dim Root As Object

    Dim field As Object
    Dim value As Object

    Dim nob As String

    Dim strCommand As String

    If UserOwns(strProductName.Text) = False Then
        StatusBar1.Panels(1) = "You cannot delete this table because it is owned by " + dProductOwners.Item(strProductName.Text) + "."
        Exit Sub
    End If

    Set xmldoc = CreateObject("Microsoft.XMLDOM")
    xmldoc.async = False
    xmldoc.validateOnParse = True

    URL = "http://dmdweb.de.sc.ti.com/cgi-bin/broker?_service=beta&_program=programs.dprod_delete.sas&_debug=000" + " &userid=" + gloUserId +
    "&ver=" + gloVersion + "&product=" + strProductName

    xmldoc.Load (URL)

```

```
Set myErr = xmlDoc.parseError
If (myErr.errorCode <> 0) Then

    MsgBox ("Program: programs.wmap_distincts_xml.sas failed to execute on Unix")
    URL = strCommand

    Exit Sub
End If

Show_Free_Devices

End Sub
```

Conclusions

The current method of retrieving data via prewritten SAS programs, web-based calls and returned XML is reasonably effective, and moderately, if not highly, responsive.

As this architecture evolves, the access will be faster and the system will provide more data from more sources. Some improvements that we envision are:

- Base SAS programs will be rewritten in SCL in order to provide greater speed, and to build a set of reusable classes to describe our basic set of objects.
- We will investigate other XML technologies such as SAX, in order to allow a VB program to parse XML files that are too big to fit into the VB program's workspace.
- We might explore the use of Java and Java-based tools to implement JDBC-style access to our many Oracle databases.
- We can investigate the value of SOAP technology, a means of providing basic computational services through web technology.

The dizzying pace of emerging technology pretty much guarantees that the future of our technology will only slightly resemble the past. We are sure, however, that the SAS Institute family of products will continue to serve us well.