Paper 288-25

Messaging Systems: SAS® Tools for Internet-Based Communications

Gregory S. Barnes Nelson STATPROBE Technologies Cary, NC

ABSTRACT

Programmers have developed a variety of communication methods for disseminating information programmatically to a person or group of people. From the early "form-letter" days using PROC FSLETTER to current methods of personalized e-mail and Internet based communications, we have sought to create relevant content for timely information. In this paper, we will explore three different categories of messaging: store and forward, publish and subscribe and remote procedure calls. Each of these will be presented along with methods for constructing and implementing the message-based procedures in SAS.

INTRODUCTION

The Internet has dramatically changed the way we work. From what we do, who we talk to and how we go about doing it. Largely, it is a matter of what gets noticed. Pitney Bowes conducted a study and found that on average, American workers send and receive thirty-size messages a day. With each message consuming a scant three minutes, we can spend up to two hours a day processing e-mail. If you are like most IT professionals you send and receive a 100-plus e-mails daily – and that doesn't begin to cover the information we receive from other sources: magazines, flyers, letters, web sites, etc. – all trying to grab our attention. And now, with wireless applications on the verge of explosion, we can only expect our daily routine to continue to be consumed by information.

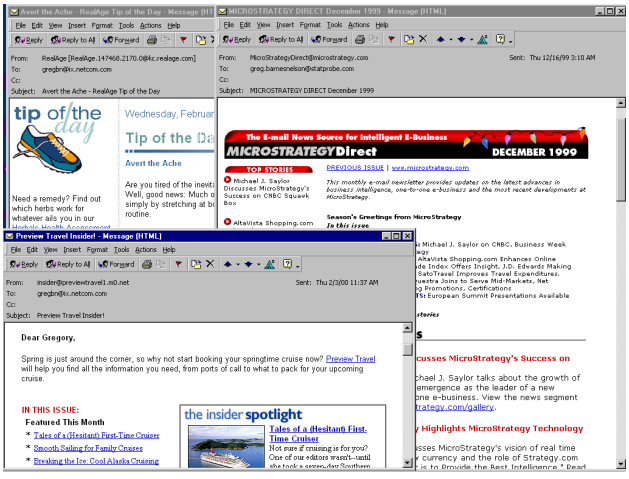

Figure 1. Personalized Web-mail.

It is likely that as a programmer, you have produced some report or assimilated some information to find that the person who requested it never even looked at it. If it is true that it is all about getting noticed, then how do we get information into the right hands and the right time?

Wouldn't it be nice if we could create compelling content for our internal users? Messages that showed people just how good the numbers were and kept them coming back for more?

The theme of this paper can be characterized as "making information matter". Instead of a traditional marketing approach, we look at things in this paper from a technologists' perspective. This paper is about getting information to the right people at the right time. Specifically, we will explore the broad category of messaging and the tools available to us within the SAS System to help us get information noticed.

MESSAGING DEFINED

Messaging falls into the broad category of "information exchange". Messaging, by definition, includes the methods and processes that we use to exchange information – structured or unstructured data – between people and/or applications. It is about getting information from a source to a destination. There are a variety of messaging applications that help us communicate within and between components of an overall information exchange process, here we will define three categories of messaging: store and forward, publish and subscribe and remote procedure calls.

By using this framework, we will explore how information can be exchanged primarily between applications that people ultimately use. Specifically, we will focus on tools and technologies that allow information to be programmatically routed to people using the SAS System.

Store and Forward

Store and forward is perhaps the most common type of messaging and encompasses electronic mail as well as asynchronous message queues (more on message queues later.) E-mail provides the simplest example: someone creates a message (using an e-mail client), packages it up in an e-mail message and sends it off to one or more people. In this case, the message has a common format. It contains a "From" address, a "To" address, possibly a "Subject" and the message content. The content can exist of plain text, some markup text (e.g., HTML) as well as attachments.

Store and forward applications take advantage of the fact that messages can be constructed in one place (using an application like Microsoft Outlook or Lotus Notes) and deconstructed in another place. The conduit for this communication is SMTP (Simple Mail Transfer Protocol) and is well defined as an industry standard. That is, both the sender and receiver

understand the format of the message and how to encode and decode the message. Store and forward applications take advantage of the fact that the message is understood on both ends.

Publish and Subscribe

The second category of messaging is popularly referred to as **Publish and Subscribe** but is often referred to as push or distributed events services in the industry. Publish and subscribe delivers a specific message to a group of people that have "registered" themselves to receive it. The subscriber (or consumer of information) tells the "publisher" that they are interested in a particular topic and want to be notified once the information has been updated. The publisher acts as a broker, in a sense, in that it waits or "listens" for a program to tell it that some information has been published. Once notified, the broker (publisher) sends a message to the user letting them know of the update.

The message that is brokered between the publisher and the subscriber can take on several forms. The most common implementations of Publish and Subscribe include push technologies and channels, as defined by the Channel Definition Language (CDL). An example of push would include the popular application, PointCast. PointCast was an early entrant into the push wave because it could deliver news, stock quotes and other content auto-magically throughout the day. The users simply had to tell PointCast what they were interested in, then sit back and watch.

Push has fallen out of favor in most organizations lately because of the high-network bandwidth that was consumed by subscribers getting continual feeds of information. The proprietary format of the feed became a deterrent, as it required the subscriber to install, configure and register yet another application that had to be supported through their corporate IT departments.

Remote Procedure Calls

Remote Procedure Calls or RPC extends the messaging concept to include a range of technologies that allow for inter-application communication. As in most programming languages, we have the concept of a function or a method call. These constructs allow us to partition some of the application logic into reusable code segments and usually perform some very specific task such as calculating and returning a value. Remote procedure calls extend this idea so that applications can essentially share "functions". This allows a remote computer over the network to actually perform the work and return the results to the calling machine.

SAS TECHNOLOGIES FOR MESSAGING

The SAS System is a powerful suite of products that span industry types and users. Long-time SAS programmers and developers value SAS for its robust end-to-end data management features. These have traditionally included procedures to access, transform, integrate, analyze and report data. By extending SAS' core competencies, we can use the SAS System for the distribution and notification processes integral to information management. Here we will explore these using the framework that we described above: store and forward, publish and subscribe; as well as remote procedure calls.

STORE AND FORWARD

We described store and forward earlier as a mechanism to get information from one point to another. The strength of store and forward is that the sender can initiate a message (e.g., e-mail message) without too much worry about whether or not the recipient is listening. With e-mail, SMTP protocols manage how often the e-mail should be sent, how often it should try to contact the server and what to do with the message if a valid recipient cannot be found. With message queues, the message will be queued (or stored) until it receives a "I'm ready" message. Here we describe two types of store and forward messaging: e-mail and message queues.

Electronic Mail

Starting with e-mail is the simplest because so many people are familiar with it. In SAS, there are a variety of ways to construct, send and even process e-mail messages. Most of what we will discuss here is how to do this in unattended mode. In Version 8.0 of the SAS System, users now have the ability to send e-mail directly from within SAS and attach the current contents of a SAS window. This is done through the interactive graphical user interface (GUI).

Unattended mode, however, relies on the programmatic processing of SAS code to construct, send and process e-mail. We will discuss two methods for sending email using some of the basic tools within SAS (BASE, SCL) and then show two interactive examples using SAS/IntrNet and a new Version 8 feature bundled with SAS Integration Technologies: SAS Publisher.

By using the power and flexibility of Base SAS and SCL, we can create personalized messages, subset the a list or database of recipients and provide notification methods for our SAS jobs (completion codes, etc.)

BASE SAS

In Base SAS, we can use either the email device type or the PIPE device type on the Filename statement to send our email.

The PIPE device type can be used in UNIX environments to redirect the contents of the fileref to a program. Here we use the mail program instead of writing to a specific file. A simple example is given here.

```
filename mail pipe 'mail gregbn@ix.netcom.com';
Data .... /* Data Step that writes to the mail fileref */
```

For a general-purpose approach for UNIX and non-UNIX environments, we can use the filename statement to construct and send our email. By using

the MAIL device type on the filename statement (or the FILE statement within a Data Step), we can specify options such as email recipients, subject line and any attachments that we may want to include. We would then us PUT statements to create the body of the message. Here the power of SAS programming logic can programmatically create whatever information message we want.

The following macro can be used to send a simple email message.

```
%macro mailme (address=, subject=, attachment=);

filename mail EMAIL;
data _null_;
      file mail;
      put '!EM_TO!' "&address"
        / '!EM_SUBJECT!' "&subject"
        / '!EM_ATTACH!' "&attachment"
        / 'Hi Greg.  I have attached my config file'
        / '!EM_SEND!'
        / '!EM_NEWMSG!'
        / '!EM_ABORT!';
run;
%mend mailme;

%mailme(address=gregbn@ix.netcom.com, subject=Hello
Greg, attachment=e:\sasv8\Sasv8.cfg)
```

Code Segment 1. Data Step email example.

When we run this program, the file gets attached and is sent off. The message is shown below in Microsoft Outlook.

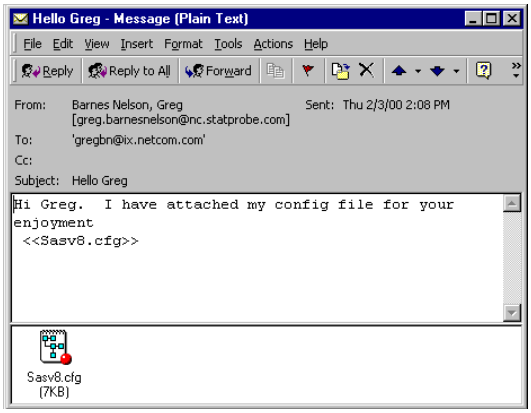

Figure 2. Message sent with the SAS Data Step.

This is obviously a very simple example, but demonstrates the power of using the Data Step for programmatically sending email. If used in conjunction with CALL EXECUTE, for example, we would build the macro calls from a database of email addresses. The downside of this approach is the continuous opening and closing of the SMTP port – that is, messages are send one at a time, in sequence, and don't take advantage of modern SMTP queuing. For very large lists (over 1000 email addresses), one would be better off creating a system that queues the messages and groups messages by domain (the server to which these messages are being sent).

In the section on SAS/IntrNet below, we demonstrate the use of a queuing method to send email through a web-based interface (see Figure 3 below).

SAS Component Language (SCL)

SCL, or SAS Component Language, is also a very powerful language within the SAS System that can be used to programmatically construct and send email. Similar to the Data Step example above, we use the filename statement and write to it using the mail device type. Instead of using put statements, we use the fput statement.

```
sendmail:
 rc = filename('mailme','userid','email');
 file_identifier = fopen('mailme','o');
 if file_identifier > 0 then do;

  filerc1  = fput(file_identifier,
       "Hello Greg.  Here is the file.");          Message Body
  rc = fwrite(file_identifier);

  filerc2  = fput(file_identifier,
     '!EM_TO! '||"gregbn@ix.netcom.com");          Email To
  rc = fwrite(file_identifier);

  filerc3  = fput(file_identifier,
                '!EM_ATTACH!'||
                "e:\sasv8\Sasv8.cfg");            Attachment
     rc = fwrite(file_identifier);

     filerc4  = fput(file_identifier,
           '!EM_SUBJECT! '||
           "The file you requested");             Subject Line
     rc = fwrite(file_identifier);

     closerc  = fclose(file_identifier);
end;
return;
```

Code Segment 2. SAS Component Language email example.

This example shows only the SCL portion. We could create a FRAME entry that could be used to drive the parameters (the user ID for the send to, the user ID for the CC, the name (if any) of a file to attach, the subject of the mail, and the body of the message.

Moreover, SCL can be used in the development of a mail object using object-oriented techniques available in SAS/AF. Here we could create a mailto method that could be called as needed. By creating an SCL Class, we could call the object from with SAS, from web pages using SAS/IntrNet or even from Java and

Java Server Pages applications (using AppDev Studio).

Reading E-Mail

In addition to constructing and sending email from within SAS, we can also read and process email programmatically. For example, by using SAS/Access, we could use the OLE DB data providers to access mail programs such as **Microsoft Exchange** or **Microsoft Outlook**. Because Microsoft has developed an OLE DB jet engine to allow these mail programs to be accessed by third party programs, we could use this to process information programmatically through Data Steps or Proc Steps.

Secondly, on UNIX systems, one could implement an email processor that takes advantage of **ELM** (a UNIX mailer) and its filtering capabilities to redirect email to a SAS process. The SAS process then parses the subject line and contents if needed. This was used in the development of the SAS File Contribution Server at the University of Georgia. See Langston and Nelson, 1995 (Langston and Nelson, 1995) for more information on this resource.

SAS/IntrNet

SAS/IntrNet represents a suite of products that allow developers to create flexible decision support systems on the web. With SAS/IntrNet, we can use the power of both BASE SAS and SCL behind web-interfaces to create interactive applications. Similar to the email example we showed above, we can create a front-end to our SAS processes using a combination of technologies such as HTML, Java and JavaScript.

The screen shot below (Figure 3) shows an example of an application that we built which builds email messages dynamically and then places them in a drop directory, which is monitored by a mail agent. Whenever a message appears in the drop directory, a monitor picks up the message and delivers it. In non-queuing applications, there can often be a bottleneck when the mail agent has to wait for an open SMTP port in order to send the message. By using a queuing model, we can release SAS from having to wait until the message is delivered before continuing.

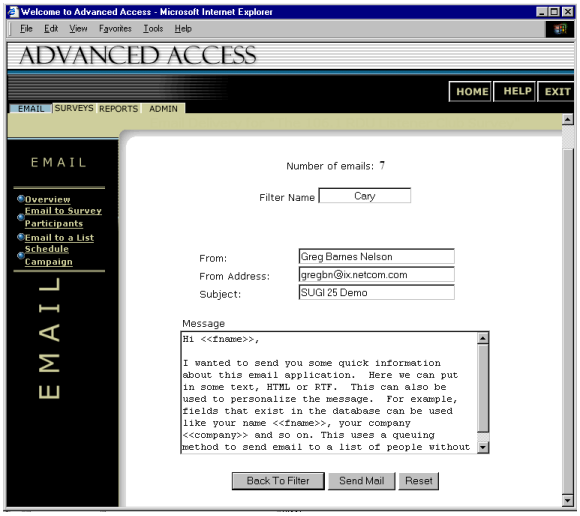

Figure 3. Web-based email messaging.

SAS Publisher

Finally, to round out our discussion of electronic mail, SAS Institute has release a suite of products, known as SAS Integration Technologies, which enable developers to integrate other technologies with the SAS System. One component of this suite is the SAS Publisher. The SAS Publisher lets you create "packages" that you can send to whomever you want. Packages can consist of SAS datasets, MDDB files, SAS programs, HTML files, etc. and can be sent to a list of email recipients, a channel (more on this later) or a message queue.

By using SAS procedural statements, we can construct SAS packages built using the SAS Output Delivery System (ODS). Once built, we could use the integrated object model (IOM) to programmatically send the output to a variety of resources (email, message queues and channels.)

Message Queues

Message queues represent the second type of store and forward messaging. Applications that run on two or more computers within a network can communicate with one another by passing messages between them just like you did in fifth grade. When a message is passed between applications in this manner one of three things can happen:

- Application 1 sends a message and waits for a response from Application 2.
- Application 1 sends a message and expects a message in return (from Application 2), but continues doing what it was doing.
- Application 1 sends a message and doesn't expect a message in return from Application 2.

The first scenario is referred to as synchronous communication (or message passing communication). This requires that both applications are up and running at the time the message is sent. A phone conversation or an HTTP request from a

web browser to a web server would be examples of this kind of communication.

The second and third scenarios above represent asynchronous communication. In these cases, application 1 doesn't care whether or not application 2 is listening because it has passed off the work to a broker, or message queue.

Message queuing is a middle-ware technology that facilitates reliable communication between applications. One benefit of this type of communication is that the sender can initiate its message and continues on with its work without waiting for a response from the receiver of the message. The message goes into a queue and waits until the recipient is ready to process it. This is also referred to as a loosely coupled communication model because the two applications are independent of one another.

Other benefits of message queuing include:

- **Guaranteed delivery.** Message queues can contain verbose logging to ensure that the message was delivered and often provide fail-safe measures for recovery.
- **Messages can be routed.** Messages can be stored and forwarded at a later time if there is no network transport mechanism between applications. Applications aren't affected by fluctuations in network speed and availability.
- **Messages are handled as transactions.** Because messages are essentially transactions, commits and rollbacks can be performed based on the outcome of the message delivery.
- **Communications can be handled off-line.** Since the queue doesn't rely on a persistent network connection, messages can be queued locally and sent at a later time.
- **Priority-based.** Messages can have a priority. The higher priority messages are delivered and routed first.

Support for Message Queues in SAS

Up until now, there hasn't really been a facility within the SAS System that could support the type of communication with a message queuing facility. With the Nasvhville Release of the SAS System, particularly with Version 8, we are just being introduced to two methods for interacting with messaging systems. The first is bundled within SAS/Connect and the second is included with SAS Integration Technologies.

SAS/CONNECT

SAS supports both synchronous and asynchronous messaging through SAS/Connect. Asynchronous (or indirect-messaging) allows developers to write applications that communicate indirectly. That is, one application can send a message to a second message without concern as to whether that second application is ready. SAS message queuing enables us to do this by placing our messages in a queue for storage. The SAS message queue manager is part of the DOMAIN server (which is a component of SAS/Connect.)

The interface to the message queue is handled through an SCL interface and through a SAS Data Step or a macro call. Through these interfaces, we can send, query and receive information from our message queue. SAS Message queuing is available in Version 7 and beyond.

SAS Integration Technologies and Application Messaging

SAS Integration Technologies is a suite of tools that help open SAS up to become more accessible through well-defined, industry standards. SAS Integration Technologies currently supports **IBM's MQSeries** and **Microsoft's MSMQ** messaging systems. This feature allows developers to create applications with SAS to exploit enterprise-wide messaging systems so that information can be exchanged with other applications. Using messages and queues, we can create interfaces for sending and receiving information between applications.

Applications built with a messaging queue offers ease of development, greater reliability, a high degree of compatibility with various systems, and other features that are essential to intersystem communication. The message queue is responsible for keeping track of machines and queues dynamically in a centralized directory service, improving scalability by not requiring changes to individual machine configurations. In addition, applications on any machine within a message queue can send messages to an application on any other machine without requiring pre-configuring channels or routes. This dramatically improves scalability by eliminating management tasks that increase exponentially with the number of connected machines. For more information on message queuing with SAS Integration Technologies, refer to http://www.sas.com/rnd/itech/doc/messageq/index.html

PUBLISH AND SUBSCRIBE

Although publish and subscribe is not a new technology, its promises have been long overdue. As mentioned earlier, publish and subscribe is a mechanism that allows people, or "subscribers" to sign-up to receive some information. For example, the following screen shows a quick way to get people to sign up to receive specialized information via email.

Figure 4. Web interface to subscribe to an e-mail service.

Using those techniques described above when we talked about producing email from a SAS Data Step or SCL, we can use this web interface to keep track of what people are interested in and notify them when appropriate. This is akin to publish and subscribe.

Push

Push (or "server-push") is the when information is delivered to a user (via their client) but the request was initiated by the server rather than the browser or the client, as is usually the case. Earlier, we gave the example of **PointCast**, which was a service that specialized in "pushing" information rather than having it "pulled". **Marimba** is a somewhat similar site (and product) that pushes information to the user on a predefined schedule.

Usually, information that is pushed from a server to a user actually comes as the result of a programmed request from the client in your computer.

Another form of "pushed" information is e-mail. Although the e-mail client in your computer has to occasionally go to your local e-mail server to "pick up" the e-mail, the e-mail arrived because someone sent it (pushed) it to you without a one-for-one request having been made.

SAS/IntrNet

In a corporate intranet, we could use this technique to notify people when reports are ready or create a customized home page for users based on their userid and group privileges. The screen presented below is part of an application that was built to allow users to be notified when a document, or resource, has been updated. Users then have the option of checking which reports they want to appear on their home page.

Figure 5. Personalization built with SAS/InrtNet.

Despite the dynamic nature of this application, it is not considered true “push” as the client (user) has to make the request to the server to get the information. By using email notification to let the user know that the reports are available and a customized interface, we have achieved one of our goals: getting our message noticed. Whether or not the user actually clicks on the hyperlinks and previews the reports is yet another challenge.

In the above examples, SAS code was used to dynamically build both the user interface as well as process the personalization of the home page and provide the email notification. This was done with Base SAS code in conjunction with **SAS/IntrNet**.

XML and Dynamic HTML

XML, or the eXtensible Markup Language, has tremendous potential for application developers – especially as it relates to the delivery of information over the Internet. In a related paper (Barnes Nelson, 2000), we show you how to create an XML document from a SAS data set. In the example below, we have taken a SAS data set (SASUSER.CLASS) and sent it scrolling across the top of our HTML using the marquee object. The marquee object takes text as its input and allows us to create scrolling messages that appear on our web page.

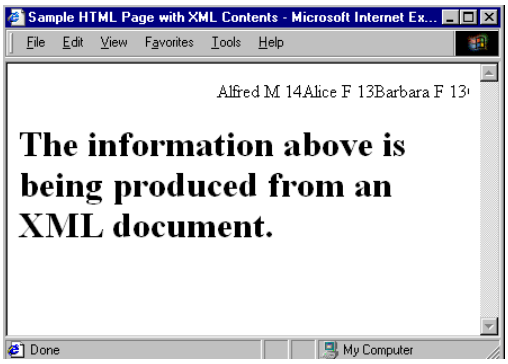

Figure 6. XML document delivered through HTML.

By extracting the data (XML document) from the way that it is being presented (HTML), we can update our marquee object as often as we need by sending the object a fresh command programmatically. The code for parsing the XML document and sending it scrolling across the screen is given below.

```
<html>
      <head>
      <title>Sample HTML Page with XML Contents</title>

<script LANGUAGE="JScript">
      var xml = new ActiveXObject("msxml");
      xml.url = "http://localhost/xmldemos/Roster.xml";

      var ClassRoster = xml.root.children;
      var string = " ";
      for (var i =0;i<ClassRoster.length;i++)
      {
        string += "<SPAN>" +
          ClassRoster.item(i).children.item(0).text +
           " " + ClassRoster.item(i).children.item(1).text +
           " " + ClassRoster.item(i).children.item(2).text;

        string += "</SPAN>";
      }
</script>
```

```
<SCRIPT LANGUAGE="JScript">
function loadData() {
      document.all("Roster2").innerHTML = string;
}
</SCRIPT>

<BODY onload="loadData()">
  <marquee ID=Roster2 width="100%"></marquee>
  <H1 id=Roster>The information above is being produced
from an XML document.</h1>
</body>

</html>
```

Code Segment 3. HTML for converting and displaying an XML document in a web browser.

The key to this code is:

1. We instantiate the XML document (roster.xml) and begin to parse it using the built in Microsoft XML parser in IE 5.0 {ActiveXObject("msxml")}.
2. Once parsed, the variable string contains a text string consisting of the tag with three variables out of the SASUSER.CLASS (Name, sex and age) dataset.
3. Finally, the dynamic HTML call to innerHTML sets the value of the marquee object to the value of string.

By using DHTML in combination with XML, we can create very dynamic applications that can take on "push" characteristics as we can update the XML document as often as we would like. By using a timeout parameter on the marquee object, we can provide continual updates to the browser without having the user do anything.

SAS Collaboration Server

Another technology that provides a publish and subscribe model is the **SAS Collaboration Server**. In partnership with **Intraspect**, SAS has created a knowledge management tool that allows information to be stored in an object-oriented database. Once stored, it can be searched, retrieved and displayed. As documents are added to the repository, users can be notified via email or through their home page of the new/ modified information. Users can subscribe to areas, or cabinets, of interest and be notified in one or more ways when information changes.

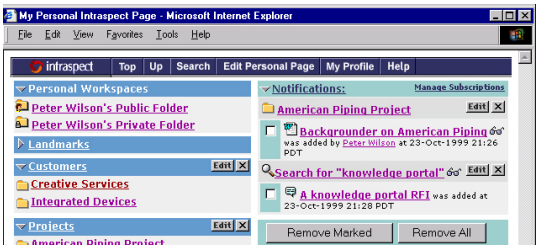

Figure 7. SAS Collaboration Server - Personal Home Page.

The SAS Collaborative Server can be used to notify users of a document change, a discussion group contribution and even when data changes on dynamic web pages built with SAS/IntrNet.

Channels

Channels (a variant of push technology) are a category of technologies that provide for the "push" of information to a computer user's desktop interface through periodic and generally unobtrusive transmission over the Web. Channels are provided through a web browser such as Microsoft Internet Explorer and Netscape's Netcaster.

In a typical intranet scenario, a web browser that supports channels is installed on the end user computer. Depending on the product, a range of provided program "channels" may be available to provide the intranet users with international, national, and perhaps local news or news headlines and possibly other services. The intranet manager decides which "channels" to preselect for intranet users and which channels (or whether there will be any) available for the user to choose.

From a corporate perspective, the value of channels comes from the delivery of corporate-relevant information to the desktop. Organizations can develop new channels to put on the server that will "push" corporate news, industry or trade news, and news about competitors to selected users in the company.

Channels can be built using a variety of middle-ware technologies such as Microsoft's Active Server Pages and web classes or in Java using SAS Institutes' AppDevStudio to construct Java servlets or Java Server Pages. Content can then be personalized using information stored about a person or group.

SAS Integration Technologies

Although anyone can create a channel using the Channel Definition Language (CDL)[1], the difficult part with channel technology is often creating, managing, publishing and archiving content. In addition, managing subscriptions and overall channel administration can be manually intensive work. SAS Integration Technologies provides facilities for managing subscriptions through the **SAS Subscription Manager**. Here, administrators can

[1] See http://support.microsoft.com/support/kb/articles/Q174/0/55.asp

manage users and groups and which resources (channels) people have access to.

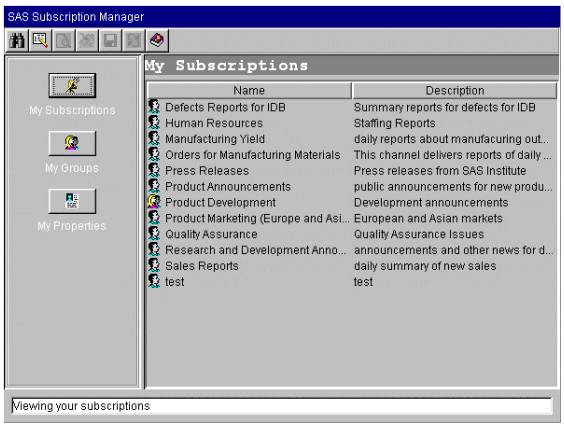

Figure 8. SAS Subscription Manager.

Publishing is done through the **SAS Publisher**. Here information can be published through the SAS interface or through the SAS Data Warehouse Administrator. As was mentioned previously, data sets (SAS data sets, MDDB, FDBs, etc.), SAS output, HTML and other documents can be published to email, message queues or channels. The Subscription Manager and SAS Publisher interface with enterprise directory services such as LDAP to surface user and group information. LDAP, or Lightweight Directory Access Protocol, can be exploited through the SAS Integration Technologies to manage publish/subscribe channel and subscriber profiles.

REMOTE PROCEDURE CALLS

To round out our discussion of message types, we can use remote procedure calls, or RPC, to create compelling interactions between our applications and our users. As mentioned previously, RPC can be thought of as distributed functions – functions that can be called from one machine and executed on another. From a SAS perspective, we have had the capability to do this with SAS/Connect for quite some time. In Version 7, we were introduced to asynchronous remote submits which allowed programs to continue running while a second machine did its work.

In SAS, we can use the idea of SAS macros or SCL methods (or even objects) to segment our code into a nice, neat little package. If we had SAS/Connect, we could even run that macro on a second machine with little trouble. With Remote Procedure Calls, we can extend this model – but instead of using SAS/Connect, we use a bridge of sorts to communicate with a remote session.

Stored Processes

Beginning with Version 8, we are introduced once again to SAS Integration Technologies and a feature called stored processes. Stored processes are a facility that enables SAS source programs to be stored on a server and executed by a client. In this scenario, a client could call a SAS stored process on a server, by name and pass it some values similar to how a web form passes parameters to a CGI program. That is, the values are formatted as a series of name/value pairs.

Stored processes can be called from both SAS clients as well as non-SAS clients. Non-SAS clients currently include COM and JDBC clients (e.g., VB, VBA, VBScript, Delphi, C++, J++ and Java). In the next release of SAS Integration Technologies, CORBA clients will be supported making SAS server components more open to industry standards.

By exploiting SAS as a server component, messages can be initiated from both SAS and non-SAS clients to extend current applications' reach throughout the enterprise.

CONCLUSION

In this paper, we have discussed three types of messaging: store and forward, publish and subscribe and remote procedure calls. Each of these messaging types can be combined to create a framework that can be fully exploited by various products within the SAS System. As we have seen, we can use some of the more fundamental tools within SAS to create customized email messages, we can use SAS/IntrNet to create dynamic interfaces to corporate information and SAS Integration Technologies to create true enterprise wide architectures that consist of a wide variety of applications.

By combining these technologies, our goal of "making information matter" can be achieved by applying the right tools to the right job.

BIBLIOGRAPHY

Barnes-Nelson, G.S. (1999) "Extending the Life of Your AF Application: Exploiting the Model-Viewer Paradigm," *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference, Miami Beach, FL.*

Barnes Nelson, G.S. (April, 2000) "XML and SAS: An Advanced Tutorial". *Invited paper presented at the annual convention of the International SAS Users Group (SUGI), Indianapolis, IN.*

Garner, C. (1999), "How to use Version 7 to Optimize the Distributed Capabilities of SAS Software", *Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference,* 24, 268-277.

Gass, M., Jenisch, S. and Kelly, C. (1999) "SAS Integration Technologies Overview", *Paper presented at the annual convention of the International SAS Users Group (SUGI), Orlando, FL. (see also: http://www.sas.com/rnd/itech/papers/oviewSUGI24.html)*

Langston, Richard D. and Nelson, G.S. (April, 1995) "Introducing the Sample Library File Contribution Server", *Paper presented at the annual convention of the International SAS Users Group (SUGI), Orlando, FL.*

Shoemaker, Jack (1997) "Let's Not Forget E-Mail", *Proceedings of the Twenty-Second Annual SAS Users Group International Conference,* 22, 878-83.

CONTACT INFORMATION

The authors may be contacted as follows:

Gregory S. Barnes Nelson
STATPROBE Technologies
117Einburgh South, Suite 202
Cary, NC 27511

Internet: greg.barnesnelson@statprobe.com
Personal: gregbn@ix.netcom.com
Web: http://www.statprobetechnologies.com

For the latest version of this paper, please refer to:
http://www.statprobetechnologies.com/downloads