ABSTRACT
With the advent of the Internet and the increasing mobility of its users, it has become necessary to allow immediate access to data and reports from non-desktop (wireless) devices such as Personal Digital Assistants and phones. In this paper we will show how SAS technology will keep your users connected to their business information anywhere and at anytime.

INTRODUCTION
The face of business is changing and corporate executives and managers must be mobile in order to meet the fast-paced changes in customer and market demands. To meet these requirements, executives are turning to wireless devices, such as the cell phone and Personal Digital Assistant (PDA), to remain in touch with the office while attending to customer needs.

The Cahners In-Stat Group forecasts that the number of non-desktop (wireless) users will surpass 1.3 billion by 2004. Additionally, it is projected that more than 1.5 billion handsets, PDAs, and Internet appliances will have wireless capability by the end of 2004.

Wireless communications and the Internet are becoming increasingly intertwined. Using the combination of SAS and the SAS technologies described in this paper, the mobile professional can have their information delivered wirelessly to a cell phone, a PDA or other non-desktop device.

Mobile professionals spend a large amount of their time away from their office where they are connected. When they are in the office they are plugged into the network and can access all of the systems, data, and applications they need to perform their duties. Many other professionals, i.e. the rest of us, also spend time away from our desk where we too are plugged into the corporate resources. What happens when people are away from the office and they are unplugged? They have to find a network port if they are in the office, a phone jack if they are on the road, or delay making the decision or getting to their corporate information until their can either find a port or get back to their desk.

Wireless technology can help to solve this problem with carefully written applications. Anyone can access their corporate information exactly when they need to from their favorite wireless device - be it a PDA, a digital cellular phone, or a two-way pager. Connecting these applications to SAS allows your users to utilize the data and compute services of SAS from a wireless application allowing you to give your users the power to know while they are on the go.

WIRELESS TECHNOLOGY
Things can get quite confusing when looking into writing wireless applications. Which device do you choose? Which markup language do you use?

There are a couple of competing standards with the two most popular standards being Wireless Application protocol (WAP) and iMode. The wireless application protocol (WAP) was created by a set of wireless handset companies that formed the industry association called the WAP Forum. The WAP Forum has since grown to over 500 members with the goal of bringing together all vendors of the wireless segment to ensure product interoperability and growth of the wireless market. To date, WAP has been most successful in Europe although it is deployed in the USA as well. SAS is a member of the WAP forum.

IMode was created by the Japanese company NTT DoCoMo and is widely used in Japan. NTT DoCoMo has formed a partnership with AT&T in the US and will be offering service for iMode in the US.

In the US some of the wireless devices support proprietary protocols to handle the wireless internet communications between the wireless device and their gateway server.

There are also the two-way email pagers that support the wireless internet such as the RIM Blackberry devices, the Motorola PageWriter, and others. The common point across all of these devices is the general architecture that they support. As depicted in Figure 1, all of the wireless devices communicate with a digital cellular tower that forwards the request to a gateway server. The gateway server reformats the message received and sends it to the appropriate web server using the HTTP protocol. The web server can then utilize current web technologies, CGI, Java, ASP, etc., to access data or computational services throughout the enterprise.

Figure 1 - General Wireless Architecture
The aforementioned protocols define the means for handling the request and response between the wireless device and the web resources, but how are the results formatted for display on a wireless device? The good news is that they all have micro-browsers running on the wireless devices, but the bad news is that each standard relies on a different markup language. WAP uses the Wireless Markup Language (WML), iMode uses compact HTML (cHTML), the Palm platform PDA’s use web clipping HTML, and the Pocket PC uses a subset of HTML.

SAS WIRELESS TECHNOLOGY

There are several SAS technologies that you may use to write wireless applications. They all rely on the protocols mentioned above to deliver information to the wireless devices. The current technologies that SAS has for wireless applications are:

- SAS/IntrNet®
- Output Delivery System with experimental support for cHTML, WML, and HDML
- WebAF™ WML transformation beans
- WebAF intelligent page transformation beans (iPage Beans)

Getting Ready
Before you start writing wireless applications to talk to SAS there are a couple of items that are recommended that you do first.

1. Obtain an emulator for the wireless device that your program will run on so that you may test your programs using an emulator before trying them on the actual device. For the cellular phones there are a couple of different emulators that exist. If you have a Nokia phone, then you will want to obtain their toolkit. You may download it from the Nokia web site, or if you have SAS AppDev Studio™ 2.0 then a free copy is included in the package. Otherwise determine what micro-browser is on used on your phone and get the emulator from the makers of the micro-browser. The most popular micro-browser for phones is the UP-Browser from Openwave Software. For Palm operating system devices you may get the Palm SDK (software development kit) that comes with a Palm emulator, and RIM provides a BlackBerry emulator with their SDK as well.

2. Setup the MIME types on the web server where the wireless applications will reside. Each of the different protocols uses a different MIME type header to inform the web server of the markup language that it supports.

SAS/IntrNet

SAS/IntrNet can be used to write wireless applications with just a few configuration changes to your web server environment. As explained above the requests from the wireless devices come in to the web server where the wireless application resides as a HTTP request. So just as in the web world where you would code a request for a SAS program behind a submit button in an HTML page, you would do the same for a wireless application in the appropriate markup language that the wireless device supports. SAS/IntrNet serves as the plumbing for the wireless devices to talk to the SAS services in the enterprise.

In order to have the web server recognize the requests coming in from the wireless devices, you have to register the MIME types to the web server. The web servers pick out the MIME type from the HTTP header of the incoming request. For example the MIME types associated with WAP are:

<table>
<thead>
<tr>
<th>MIME Type</th>
<th>Extension</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text/vnd.wap.wml</td>
<td>Wml</td>
<td>WML Source Code</td>
</tr>
<tr>
<td>Application/vnd.wap.wml-wbxml</td>
<td>Wmlc</td>
<td>Compiled WML</td>
</tr>
<tr>
<td>Text/vnd.wap.wmlscript</td>
<td>Wmls</td>
<td>WML Script Source</td>
</tr>
<tr>
<td>Application/Vnd.wap.wmlscriptc</td>
<td>Wmlnc</td>
<td>Compiled WML Script</td>
</tr>
<tr>
<td>Image/vnd.wap.wbmp</td>
<td>Wbmp</td>
<td>WML Bitmap</td>
</tr>
<tr>
<td>Text/x-hdml</td>
<td>Hdml</td>
<td>HDML Source Code</td>
</tr>
</tbody>
</table>

Table 1

Once you have the MIME types registered your webserver will be able to properly handle the HTTP requests coming from your wireless devices. The above values are only for WAP and for the other protocols you would have to configure the web server for the corresponding MIME types.

Example Wireless Sales Report

Let’s say that we would like to create a wireless application that would allow a user to view the current status of the sales figures by region or state on a WAP capable device. Of course the data that we will be accessing is part of a data warehouse created by SAS and we have some summary tables for the regional and state sales figures. The regional sales figures are in the regsal SAS data set, and the state sales figures are stored in the statesal SAS data set.

WML uses a concept of Cards and Decks to determine how much should be displayed on the wireless device. Each card represents one screen of information, and all of the cards from one WML page make up a deck.

The first order of business is to create the initial page that the user views when starting the application. In this simple example we are going to give the user the choice of viewing the Regional figures or the State figures.

```xml
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="MainMenu">
<p>
<br/>
</wml>
```
When the application is deployed this page will send the initial menu to the end user and will display as seen in Figure 2.

![Figure 2 - Main Menu on a WAP](image)

Using the arrow keys the user will select which report they would like to see. We would like to see the Region Report, so all we have to do is to press the button below the word OK. This tells the phone to select the highlighted menu choice and since it is a link to another document the micro-browser in the phone looks at the link and follows the URL to request the next page. In this case it is pointing to Region Report, so the micro-browser requests the associated URL:

http://www.yourserver.com/scripts/broker.exe?_PROGRAM=regsal.sas&_SERVICE=WAPService

Using SAS/IntrNet and it's functionality SAS runs the regsal.sas program to access the summarized data from the data warehouse and send back a new WML page to the phone to display the results. The first steps to complete are to create the WML header, to start the body of the WML page, and setup the the WML table with the proper header elements.

```sas
data _null_; file _webout; set salesdat.sumreg end=EOF;
/* Specify the WML header */ if _N_ = 1 then do;
  put 'Content-type: text/vnd.wap.wml'
  put '<!--<wml> -->
  put '<card id="IDX">'
  put '<table columns="2">'
  put '<tr><td>Region</td><td>Sales</td></tr>'
  end;
run;
```

Notice that the SAS/IntrNet reserved file _webout is being used to send the output back to the wireless device.

The next step is to insert the rows of the table from the regional sales figures table. Note that the data _null_ step continues with the following statement to format the rows:

```sas
put '<tr><td>region '</td><td> Total '</td></tr>;
```

The final step is to finish the WML page. By detecting the last observation (row) in the data using the end=EOF option, we can write the last set of WML statements after the above source line has written the current values out using the following code.

```sas
if EOF then do;
  put '</table>';
  put '</card>';
  put '</wml>';
end;
run;
```

When the program completes the user will see the sales figures (see Figure 3) summarized for each of the regions as represented in the data on their mobile phone regardless of the time or place.

![Figure 3 - Regional Sales Figures](image)

### Output Delivery System & Wireless

Using the data _NULL_ ; step and manually generating the WML code gives the programmer full control of exactly what is generated, but this could be simplified using the experimental support for WML in the SAS Output Delivery System (ODS). The entire program can be replaced with the following few lines of code to get the same result on the phone’s display.

```sas
Title "Region Report";
ods wml file="_webout";
proc print noobs data=salesdat.sumreg;
run;
ods wml close;
```

The examples above are written for WAP and WML, but they could also be written for the other wireless protocols and markup languages such as iMode, cHTML, HDML, etc. The SAS Output Delivery System experimental support for wireless includes WML, cHTML, and iMode.

This sample application is a simple application but its purpose is to introduce you to the SAS technologies that you may use to write applications that deliver not only data, but also may surface the results of SAS computational services from enterprise resources. The wireless markup languages support the popular user controls such as list boxes, radio buttons, check boxes, and forms. Using these controls you can create more advanced wireless web applications that utilize the wireless device as the user interface and the SAS Server as the computational resource. The selections made by the user are passed to the SAS program as macro variables allowing the SAS programs to
generate the dynamic content that the user is requesting.

**JAVA TECHNOLOGY**

Java is a software language that is platform neutral. It is said that Java is “Write Once, Run Anywhere ™”. That is, an application developed in Java can be deployed to any machine that supports the Java™ Virtual Machine. Two important technologies that build on Java are Java™ Servlet and JavaServer Pages ™ (JSP ™).

When used in conjunction with wireless technology, developers can use their knowledge of these technologies to provide dynamic delivery of data such as WML. Java’s component-based technology, JavaBeans™, makes it easier to build web pages using JSPs and Servlets. JavaBeans separate the user interface from the application logic. This enables the page designer to focus on writing the presentation layer while allowing the application developer to generate the dynamic content portion of the page using Java and JSP. Additionally, JavaBeans provide an integration standard. This is important for developing solutions in heterogeneous environments within the enterprise or across the Internet. SAS Institute Inc. adheres to this standard and offers component-based JavaBeans that allow easy access to complex SAS resources.

**SAS TECHNOLOGY**

SAS Institute Inc embraces Java technology. webAF, the first Java-based integrated development solution to be tailored for the information delivery environment, is a complete suite of application development tools for building thin-client Java applications. Applications written using webAF can tap into SAS resources through Java technologies.

The component technology in webAF makes it easy to use a standards-based approach to access SAS from the Web. webAF software is a Java framework that enables access to SAS/AF® objects, tables (data sets), multidimensional databases (MDDBs), and other SAS computing resources. webAF provides data models that enable developers using JavaServer ™ Pages (JSP ™) to create dynamic content that maximizes the capabilities of the SAS System.

The power of webAF's JSP support lies in its InformationBeans™ and TransformationBeans. InformationBeans encapsulate SAS data by presenting it as webAF data models. The webAF data models are then, in turn, consumed by TransformationBeans, which transform the model into appropriate presentations. A key to integrating SAS into wireless technology lies in the server-side processing of the HTTP request. Here the TransformationBeans display SAS data for the appropriate wireless enabled device or PDA.

**WML TRANSFORMATIONBEANS**

Java Server Page™ (JSP™) technology is an exciting new technology that many web developers are starting to embrace. The combination of JSP™ and SAS TransformationBeans are a powerful method for accessing and delivering information to people whenever they need it. As a result, SAS is delivering WML TransformationBeans using Java on the server to automatically generate WML for display on the user’s wireless devices.

The current set of WML TransformationBeans in webAF are:
- WML Table
- WML Listbox
- WML Radio button

**Wireless Sales Report Using WML TransformationBeans**

Java Server pages are becoming a popular method for implementing web applications. Let's take a look at the Sales Report demo application written using Java Server Pages and the SAS WML TransformationBeans. We will use the the WML Table TransformationBean. It will take care of communicating with the SAS Server to retrieve the data and will format the data using WML.

The JSP Wireless Sales Report program must first display the main menu on the wireless device to allow the user to select which report they would like to see. Then based on the menu item selected the next JSP page will run to access the data and return the results.

In order to create the JSP application using SAS’ AppDev Studio and the WML TransformationBeans, you would start the SAS® Java application development tool webAF. Once webAF loads select the new button, make sure the Projects tab is selected in the dialog box and enter your project information for your new project. Press the OK button when you have finished entering the information and your new project will be started with the first JSP page in the project, index.jsp, open and ready to go. For this example we will have a very simple first page.

First, a brief summarized overview of how the request for a Java Server Page (JSP) is handled by a web server.
- The web server receives a request for a URL of a JSP page: http://yourserver.com/index.jsp
- The web server passes this request to the appropriate application (sometimes referred to as a Java Web Server, or a plug-in) for processing a JSP URL request
- The first time the JSP page is referenced the JSP processor compiles it into a servlet.
- The servlet engine (usually the same as the Java Web server) starts processing the requested .jsp page and begins building the response file based on the contents of the .jsp file
- • %--are comments and are ignored
- • any HTML statements are echoed to the response file
- • % Specifications that Java language statements are included between these tags. The servlet engine will execute the Java statements.
- • Upon completion of processing the .jsp file, the resulting response file is sent back to the requesting client.

Here is a copy of the code for the main menu:

```jsp
<%@-- Copyright (c) 2001 by SAS Institute --%>
<%@-- Inc., Cary, NC 27513 --%>

<%@ Set the MIME type to WML --%>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id='main'>
<p>
<a href='http://www.yourse rver.com/salesapp/regi on.jsp'>Region Report</a>
<a href='http://www.yourse rver.com /salesapp/state.jsp'>State Report</a>
</p>
</card>
</wml>
```
From any WAP enabled wireless device you could then access your main menu by referencing the URL for index.jsp from the embedded micro-browser. For example the URL could be:

http://www.yourserver.com/salesapp/index.jsp

The same menu will display as the one in figure 2 above, allowing the user to select which figures they would like to see next. For our example let’s assume that they select the region report. As seen in the JSP code above this will cause the web server to run region.jsp. In the region.jsp file we have to perform the following steps:

1. Set the WML MIME type
2. Start the WML output specifying the first card and title
3. Create a connection to the SAS Server
4. Create an interface to read a dataset
5. Use the WML Table TransformationBean to access the data and format it using WML
6. Close the connection

The region.jsp file is listed below with each of the steps identified.

```jsp
<%@ page import="com.sas.servlet.beans.wml.Table" contentType="text/vnd.wap.wml" %>

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "+//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id='Region' title='Region Sales' >
<p>
    com.sas.rmi.Rocf rocf = new com.sas.rmi.Rocf();
    com.sas.rmi.Connection connection = new com.sas.rmi.Connection();

    // Create a new DataSetInterface model.
    com.sas.sasserver.dataset.DataSetInterface dataSet =
        (com.sas.sasserver.dataset.DataSetInterface) com.sas.servlet.util.Util.newInstance(rocf, connection, com.sas.sasserver.dataset.DataSetInterface.class);
    dataSet dataSet("sasuser.sumreg2");

    // Create the table object
    Table table = new com.sas.servlet.beans.wml.Table();

    // Set the model and point to the data set
    table.setModelInterface(dataSet);
    // Set properties
    table.setUseColumnHeadings(true);
    table.setColumnHeadingColumnFormat("<td><b>#META_LABEL#</b></td>");
    table.setColumns(2);
    // Output the table
    table.write(out);

    // Close the table and the connection to SAS
    dataSet.close();
    rocf.stop();

</p>
</card>
</wml>
```

Just as you have seen in the previous examples, you must specify the MIME header at the top of the WML page. This set of code uses the @page import directive to do just that. You also must specify the version of the WML that this page is based on.

Start the WML page and specify the first card (ie. First screen) that will display.

Create the objects for making the connection to SAS. ROCF is the Remote Object Class Factory and com.sas.rmi.Connection is the SAS Java class for creating SAS connection objects. Please refer to the AppDev Studio documentation for further information on these items.

Using the rocf and connection objects create an interface to the dataset (sasuser.sumreg2) that will be formatted in WML to send to the wireless device. Note: sasuser.sumreg2 is a demo data set put in the sasuser library for this paper only.

Create a new WML Table object using the WML TransformationBean: com.sas.servlet.beans.wml.Table, point it to the data, set the formatting options, and have the bean write out the WML for sasuser.sumreg2.

Close the connection to SAS, put the closing WML tags in, and the Java web server will send the resulting web page back to the wireless client.

This Java Server Page of our sample wireless program looks much different than the SAS/IntRNet based applications, but takes advantage of the JSP technology that is becoming very popular.

Some of the benefits of using JSP technology are that the Java Web Server (or plug-in) handles the session management for running the JSP application. The responsibility for running the JSP and it’s associated servlet lies with the Java Web server. The Java web server keeps track of which user goes with which session and all of the other relevant details.

JSP’s also allow your web applications to maintain ‘state’ from one request to another without extra programming. This becomes very useful as your applications become more complex and need to know the values from previous steps. The Java web
server maintains these values and keeps track of them for each instance of the JSP page that is being used.

**IPage TransformationBeans**

Now that you are introduced to the JSP technology, let's move onto the iPage TransformationBeans. The iPage TransformationBeans can make writing your applications much easier, especially if you would like to write your application once, yet be able to run it on a WML wireless device, a HTML wireless device, or an iMode wireless device. That is the power behind the iPage beans in the webAF component of AppDev Studio. iPage beans allow you to write your JSP application once yet have it support multiple markup languages.

The term iPage Bean refers to the fact that each of the iPage Beans are responsible for creating an entire page of markup language for a wireless device. The current iPage Beans in AppDev Studio 2.0 are:

- **iText** creates a single page of text
- **iMenu** creates a menu used for navigation
- **iForm** creates a form for obtaining input

Let's return to our sample JSP sales report application. To make our application independent of wireless devices we will use iPage beans. iPage Beans are device agnostic. As long as the device that is making the request supports one of the supported markup languages, the iPage Bean will be able to support it. Even standard HTML browsers are supported.

First, the JSP page for the main menu must be changed to use the iMenu bean as follows:

```jsp
<%@ taglib uri="http://www.sas.com/taglib/sasads" prefix="sasads"%>
<sasads:iMenu id="Main Menu" />
<sasads:iMenu iURL="RegSal.jsp" description="Region Report" />
<sasads:iMenu iURL="StateSal.jsp" description="State Report" />
</sasads:iMenu>
```

Not only is this piece of code much simpler than the non-iPage version, it also will support the various wireless devices using the different markup languages. In this example, the SAS custom tag version of the iMenu bean is utilized. Please see the AppDev Studio documentation for more information on custom tags.

When this JSP page is called by a micro-browser, the iMenu bean will determine the supported markup language and then will display a menu, as in Figure 2, allowing the user to select which report they would like to view.

Assuming the user chose to view the Regional sales figures report as before, then the data will have to be retrieved and formatted for the device. But, in order to keep our application browser independent, we must read the data first into text and then use the iPage Bean iText to write out the proper markup language for the requesting device. The new program for the region sales report is as follows:

```jsp
<%@ taglib uri="http://www.sas.com/taglib/sasads" prefix="sasads"%>
<sasads:iMenu id="Main Menu" />
<sasads:iMenu iURL="RegSal.jsp" description="Region Report" />
<sasads:iMenu iURL="StateSal.jsp" description="State Report" />
</sasads:iMenu>
```

Using the iText Bean to convert the text to the proper markup language:

```java
int rcount = dataSet.getRowCount();
int ccount = dataSet.getColumnCount();

StringWriter sw = new StringWriter(rcount);
PrintWriter pw = new PrintWriter(sw, true);
String tab = "
"

// Write out the column labels
for (i=0, j=0; i < ccount, j++){
   pw.print("\"");
   if (i > 0)
      pw.print(tab);
   pw.print(label[i]);
}

// Write each row to the stringBuffer
for (i=0; i < rcount; i++){
   row = dataSet.getRow(i+1);
   for (j=0; j < ccount; j++){
      if (j > 0)
         pw.print(tab);
      pw.print(row[j]);
   }
   pw.println();
}
```

Sections 1 and 2 are the same as the previous example and establish the connection to SAS, sets the data model, and points to the SUMREG2 data set that contains the data to display.
Section 1 contains the logic for retrieving the header and row information from the data set and converting it to a formatted string. In this section the Java StringWriter and PrintWriter classes are utilized for managing the strings, and the SAS dataset transformation bean methods are used to retrieve the variable labels and each row.

Section 2 is the smallest section of code in this program, yet it contains the most powerful piece that makes the program independent of the wireless device and browser that made the request for this JSP page. In this section the iPage Bean iText is used to format the text string that we just built into the proper markup language to display on the device. In the string we built, each row was formatted as a string followed by a carriage return. The iText Bean creates all of the markup code for the entire page sent back to the device. It creates the header, puts the text string in replacing the carriage returns with line breaks (<br> in HTML), and finally puts the closing tags for the page and sends it to the device.

This JSP page performs the same function as the previous example that uses the WML TransformationBean, but this page can be used by many more devices without having to change the code. It will support WAP devices, iMode devices, and HTML devices all from one application. This is very important, especially for the current state of the wireless market. Most enterprises have not standardized on a single wireless device or a single wireless markup language, making it important to have the flexibility to write applications once to support many devices.

**CONCLUSION**

The popularity of wireless devices capable of communicating with enterprise servers is taking off. As a result users are starting to demand access to enterprise information as they are on the go, so they make informed business decisions anywhere, anytime. The days of waiting until you get back to the office to make a decision are slipping away. More and more people will rely on their wireless devices such as PDA’s, digital cellular phones, and two-way email devices to access the enterprise information for making informed decisions.

SAS has been in the business of giving its customers the ability to make informed business decisions for 25 years. Now, with the support for wireless devices these same decisions can be made from information produced from SAS and viewed dynamically on a wireless device wherever the users happen to be.

Using the technologies discussed in this paper you can extend your current SAS/IntrNet applications, and you can take advantage of the popular Java server technologies for creating new applications to surface dynamic, wirelessly driven results from the SAS System.

**CONTACT INFORMATION**

Your comments and questions are valued and encouraged. Contact the authors at:

Scott Vodicka
SAS Institute Inc.
SAS Campus Dr.
Cary, NC 27513
(919) 531-4922 (Voice)
(919) 677-4444 (Fax)
Scott.Vodicka@SAS.com

**REFERENCES**

SAS AppDev Studio Developer’s Site

SAS AppDev Studio Developer’s Site: Wireless Technology

**ACKNOWLEDGMENTS**

SAS Institute Staff who contributed to the completion of this paper:

Bryan Boone  Pat Herbert
Corey Benson  Skip Smoak
Robert Girardin

**Trademarks**

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks of their respective companies.

Visual Basic, Windows, Windows NT, and Microsoft Windows are registered trademarks of Microsoft Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.