

A Generic Solution to Running the SAS® System on the Web Without SAS/Intrnet®

David L. Ward, InterNext, Inc., Somerset, NJ

web server to return the output of a SAS program very quickly.

ABSTRACT

Many organizations are not able to afford SAS/IntrNet but desperately need to provide dynamic web-based content to users, usually in the form of on-line reports. Sample CGI scripts that invoke the SAS system have been offered at various user group conferences, but a complete alternative has not been available. This paper will demonstrate the installation and usage of this alternative, as well as show examples of reports generated real-time in a web-browser using only Base/SAS. You will learn to create HTML forms that pass parameters to a SAS program that displays a table or graph according to what a user has chosen. You will also learn to use ODS to generate presentation-quality output.

INTRODUCTION

This paper will serve to show a number of things. First, the power and flexibility of the SAS System, specifically SAS/AF, in building complex Internet-based applications that communicate in a client-server architecture. Second, it will exhibit InterNext's product **Onyx**, including installation, administration, usage, and advanced features. Third, readers will see examples of how to build web-based SAS programs using either Onyx or SAS/IntrNet. ODS, SAS/Graph, and other techniques will be discussed.

OVERVIEW OF CONCEPTS

Before we explore the details of using SAS with the web you should familiarize yourself with some concepts pertinent to the topic at hand.

1. **Web Server:** A software application that allows users to connect to a computer and request web pages from it. Most web servers also allow users to request not only files but also the output of programs that are executed at the time the user makes the request.
2. **Common Gateway Interface (CGI):** This refers to a standard protocol that web servers use to invoke a program that a user requests. It does not actually refer to a specific programming language, but can be implemented in almost any. CGI defines what environment variables are passed to the program (usually called a script) so that it can process a user's request properly. The most popular CGI programming language is called Perl (the Practical Extraction and Reporting Language). Perl can be obtained freely and runs on virtually any computing platform.
3. **Web Page:** Refers to an HTML (Hypertext Markup Language) document usually given to a web browser from a web server. The HTML document can either exist on the web server or be the output from a CGI program.
4. **Dynamic Web Page:** An HTML document that does not exist on the web server, rather, it is generated by a program and returned to the user when requested.
5. **TCP/IP Socket:** A software object that allows one computer to communicate with another. TCP/IP sockets are how all Internet servers communicate with Internet clients (like web browsers). They can also be used to allow SAS to communicate with a

WEB-ENABLING YOUR SAS PROGRAMS

This paper will use the term "web-enabling" to refer to moving functionality that exists in Base/SAS, SAS/Macro, or SAS/AF programs or applications into an environment where it can be executed on another computer by user via a web browser. The user would presumably not have the SAS System installed, and would not specifically need it to be. You may already have Base/SAS reports using data steps and procedures that users run either through a command-line interface or through the SAS display manager on hosts that support it. Users who submit these programs typically need to be familiar with the SAS system and somewhat technical, especially if parameters need to be passed to the program or set in the code. Web-enabling your programs can afford you with several key advantages:

1. **Deployment flexibility.** You have a number of options on how to build a front end for your application or reports. You can provide a web page or Java applet that allows your users to provide input, and that content can be accessible to users across operating systems and browser types.
2. **Less training/software required.** Users do not need access to SAS software or the data on their local machines. If you have many users accessing your application, this can be very important. The web has become a very familiar environment in which computer users feel comfortable submitting information and retrieving results. By letting them use this familiar environment, you reduce the amount of training needed for your specific application.
3. **Less SAS licenses are required.** Though you do need a server license of SAS to accommodate many users querying SAS software on one machine, this will typically be less expensive than purchasing individual client licenses.

REQUIREMENTS

What software and hardware is required to build an environment to execute web-based SAS applications? You need a web server, SAS software running on either this same server or another physical machine, and a way to pass requests from the web server to the SAS System. SAS/Intrnet, the Institute's solution, uses CGI. The product that is the focus of this paper, Onyx, takes a more "open" approach, allowing communication between the web server and SAS by either CGI or other technologies including Sun's Java Servlets or Java Server Pages, Microsoft's Active Server Pages, and others. Many of these approaches are superior to CGI because they do not invoke a separate process for each request, allowing many more concurrent requests to be handled. Onyx offers a documented protocol for communication with the SAS system called the Onyx SAS Application Protocol (OSAP).

ABOUT ONYX

WHAT IS ONYX?

Onyx is a full-featured application server that can be used to control the SAS system in a unique way from any other software application, even on remote machines. Onyx fully and in a special way supports communication with the web. An advanced multi-threaded application server running in Java provides a transparent and load-balanced way to process SAS requests on any number of servers.

Onyx gives you the power to run your SAS applications using any kind of front-end you desire, whether it be web pages (HTML and a scripting language), Java applets, Client-side applications, or anything else. It can also run on any operating system that the SAS system can run in.

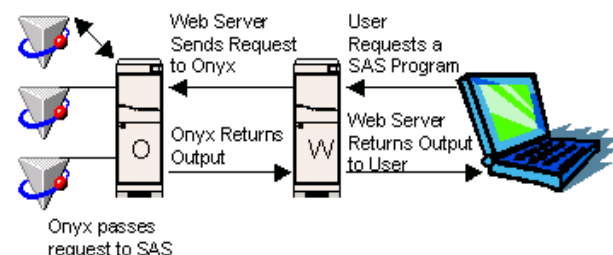
SOME FEATURES OF ONYX

- Includes an integrated, extremely fast, web server, making it very easy to develop and test SAS® code on individual PCs or laptops.
- Allows application developers to re-distribute their applications to sites that do not have Onyx licensed, giving you the freedom to require only base SAS® for your web applications.
- SAS® can be run from any number of different computers, even running different operating systems, with a load-balancing algorithm to distribute processing over the machines evenly.
- Comes with a built-in interpreter for the Onyx Dynamic HTML syntax which lets users use HTML editors to build pages that have embedded SAS® code within familiar <% %> tags.
- Is administered either via telnet or a web browser. Users can drop or add pooled SAS® sessions and check on the status of requests.
- Can easily be customized by SAS®/AF developers on the server side by adding new request types as SCL entries.
- Will run on any operating system that SAS® is licensed for.
- Requires nothing more than base SAS®.

Each of these features will be presented and discussed throughout the rest of this paper.

HOW ONYX WORKS

If you are familiar with the way in which SAS/Intrnet works, you should take special notice that Onyx has been designed with a different structure. Onyx includes a program called the "Application Dispatcher", which serves as the central nervous system and point of entry for making requests. The Application Dispatcher (hereafter referred to as simply the Dispatcher) functions almost exactly like a web server in that it waits for requests via the standard networking protocol TCP/IP, and allows a client to submit a request. In fact, the Dispatcher includes a built-in web server, which we will examine later. Here is an illustration to help you understand what happens when a user makes a request from Onyx:



This diagram illustrates how the various software applications that provide a link from the web to the SAS system function and in what order. It can also be misleading in that each step of the process is depicted as being handled by separate machines. In reality, every part of the process from the web browser to the SAS sessions can be run on one computer.

In Onyx terminology, SAS sessions that are configured to communicate with the Dispatcher are referred to as Drones, because they exist solely to do the bidding of the Dispatcher, their "queen". A significant difference between the architecture of Onyx and SAS/Intrnet arises from the fact that these Drones function as TCP/IP clients instead of servers. This means that each Drone connects to the Dispatcher and is immediately terminated if the Dispatcher application is halted. This can sometimes be beneficial to setting up a secure networking environment.

WHY SO MANY SAS SESSIONS?

You may wonder why several SAS sessions are depicted in the above diagram. The SAS System, up to the most recent version 8.2, has always been "single-threaded." This means that an individual SAS session is unable to run a procedure or data step and be waiting for another request at the same time. Nor is it able to run two separate data steps at once. This has the negative effect of requiring a separate SAS session for each concurrent request made to Onyx. If you had 10 users request a program at once, you would need 10 SAS sessions to serve these users instantly, or some users would have to wait until a SAS session finished processing the request of another user.

Unfortunately, an individual SAS session consumes a great deal of memory that cannot easily be controlled or limited without severely crippling the session. In order to allow many Drones to execute, even while each may consume a large amount of memory, Onyx allows the Drones to run on separate physical machines than the Dispatcher. In fact, you could even run Drones on machines all around the world via the Internet. The Dispatcher makes decisions about which computer and session to send a request to.

Even though you can run multiple SAS sessions that are all connected to the Dispatcher, you still may not be able to accommodate a large number of concurrent users. To solve this problem, the Dispatcher allows an option to be set that will cause users to wait for a specified length of time for an available session, after which they will receive a customizable busy message.

HOW IS ONYX DEVELOPED?

Only two programming languages are used in the development of Onyx, SCL and Java. In fact, they share so many similar features that they naturally work well together. Both SCL and Java can be compiled into operating system independent "byte-code", which allows multi-platform use. The syntax of the two is even strikingly similar. A key difference between them is the fact that Java applications can be run without purchasing Java while SCL applications require the SAS System. The simple CGI script included with Onyx is written in the ubiquitous scripting language called Perl. Perl is freely available and is installed on most web servers in the world.

WHAT INSTITUTE PRODUCTS CAN ONYX MIMIC?

Because Onyx provides a TCP/IP application server with no specific or forced client, it can be used in many different ways. The key comparison is, of course, SAS/Intrnet, but the functionality of SAS/Connect, WebAF and Integration Technologies can also be utilized with Onyx. The key feature of Onyx that enables this kind of connectivity is the Onyx SAS Application Protocol (OSAP).

ONYX SAS APPLICATION PROTOCOL

Virtually all Internet servers (web server, mail server, ftp server, etc.) use what is called a TCP/IP protocol to define the syntax of how clients and servers can understand each other. These protocols are typically created by panels of volunteers in documents called Request for Clarifications (RFCs). If application vendors use a standard protocol when designing a server, anyone can write a client that knows how to communicate with it. The same model has been used in developing Onyx. A custom, text-based protocol has been created so that anyone or anything can communicate with the SAS system in an easy and intuitive way. Such a protocol makes configuring firewalls easier as well because the syntax of messages can be anticipated.

What does an OSAP request look like? Well, it looks a lot like an HTTP request if you have ever seen one. Hypertext transfer protocol (HTTP) is the language that web browsers use to communicate with web servers. A typical request looks something like this:

```
GET /index.html HTTP/1.0
Host: 102.3.55.61
```

This instructs the web server to return the contents of the file named index.html to the host computer whose number is listed. An Onyx request looks like this:

```
OSAP/1.0 Request:
Program=/home/dward/run.sas
Session: 18931707154183480915056273279
Content-Length: 2
```

```
State=NC
Product=Onyx
```

The first line (split into two to fit in a single column) always indicates that the client is making a request and includes the details of that request (the program run.sas). The concept of sessions (on the second line) will be introduced later. The remaining lines indicate input parameters that the user wishes to send to the program run.sas.

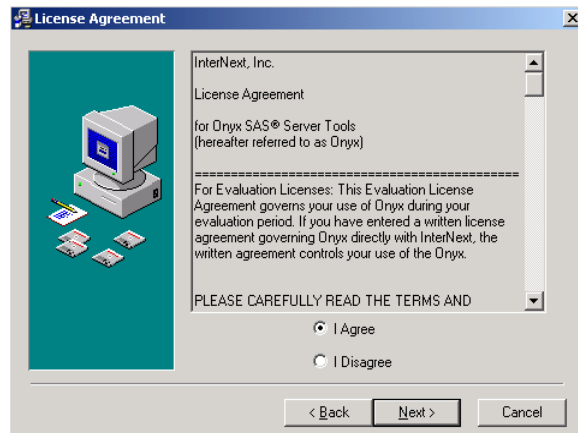
So what can you do with OSAP? Users can now telnet directly to the Dispatcher and submit requests. Onyx includes 5 built-in request types: PROGRAM (shown above), SQL, CODE, MACRO, and MACROVAR. Each one performs separate actions based on the content sent to it. The Program request type is used for web requests and would usually return HTML or graphic output. The SQL request type assumes that the content will include an SQL statement and the request details should include information about which format the user wants the results returned in. CODE allows users to submit either Base/SAS or SCL code directly to Onyx and have the results returned. MACRO allows users to request a SAS/Macro to be executed with the parameters named in the content of the request, while MACROVAR returns some or all macro variables from the SAS System. **SAS/AF developers can write custom request types that respond to OSAP requests in any way they choose, making Onyx infinitely extendable.**

INSTALLING ONYX

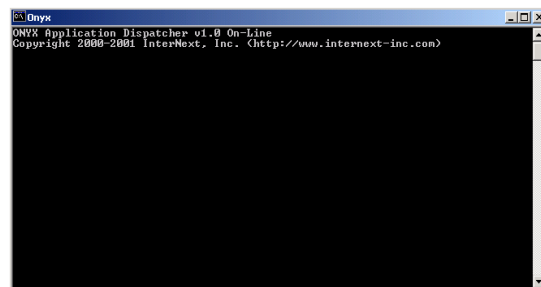
Onyx is very simple to install, particularly on Windows hosts. Simply run the Onyx installation program and follow the wizard that will guide you through setup. The installation wizard will perform the following actions:

1. Install the Java Runtime Environment, if it is not already installed
2. Copy all system files to the local computer
3. Modify configuration files with default settings and let the user change them if desired
4. Install a Windows NT/2000 service if desired

Here is a sample screen shot of the installation wizard:



After clicking on the Onyx shortcut that is placed on your desktop during installation, a command prompt window should show up in the Windows taskbar. Clicking on the window will reveal the following screen:



Once you see the message

```
ONYX Application Dispatcher v1.0 On-Line
Copyright 2000-2001 InterNext, Inc.
(http://www.internext-inc.com)
```

you know that Onyx is running. Onyx can be run even with the default settings. The test and demonstration programs will execute, all through the built-in web server. Once installed, simply navigate your browser to <http://localhost:5000/> and enter the username "admin" and password "onyx". You will immediately see the Onyx web-

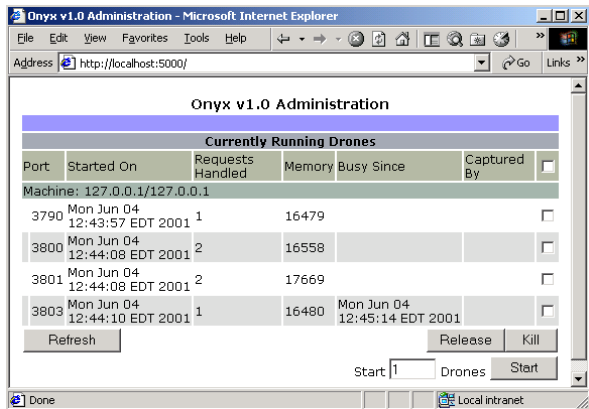


based administration utility, described in the next section.

ADMINISTERING ONYX

VIA THE WEB

Once you have Onyx installed and running, navigating your browser to `http://localhost:5000/` as mentioned above will display the Onyx administration utility. It is from this interface that you can drop and add Drones (you can only add Drones running on the same machine as the Dispatcher) and check on the status of requests. A typical screen looks something like this:



Though you can't see it well at this size, this web page indicates that there are 4 Drones available. The number of requests each one has handled and the amount of memory each uses is displayed. Also indicated is a date/time value that the Drone began processing a request on. You can use this value to see if any Drones are currently busy (as the last one is in the screen shot). From this interface you can kill or drop existing Drones, add new ones (start new SAS sessions on the machine), and "release" Drones (discussed in detail later).

VIA TELNET

If administering the server through a web browser is not acceptable to you, you can choose to use the standard telnet utility to display a similar text-based interface. A sample telnet session:

```
>Enter Onyx Admin Command:
show
>
>
>          Currently Running Drones
>
Address/Started/Requests/Memory/Busy/Captured
>=====
>Machine #0: 127.0.0.1/127.0.0.1
> #0: 1344/Fri Apr 06 13:08:57 EDT 2001/0/0//
> #1: 1347/Fri Apr 06 13:09:14 EDT 2001/0/0//
>
>Enter Onyx Admin Command:
kill 0.0/ (kill machine 0 drone 0)
kill 0.0/0.1 (kill machine 0 drones 0 and 1)
kill (kill all drones)
start 2 (start 2 drones)
start (start 1 drone)
release 0.0/ (release machine 0 drone 0)
```

In this case the administrator viewed available Drones, killed, started, and released several Drones.

USING ONYX

Enough super-techno babble. Let's get to some SAS programming! Onyx can be used to run existing SAS/Intrnet programs, or to create new programs that make use of more advanced features of Onyx. This paper will start with base/SAS programs then look at how to write SCL programs for use with the web.

BASE/SAS PROGRAMS

Our example Base/SAS programs will show how to pass form parameters to SAS, how to use ODS to generate advanced HTML output, how to use sessions, explain the special syntax known as Onyx Dynamic Html, and show unique debugging features of Onyx.

PASSING FORM PARAMETERS TO SAS

Just like SAS/Intrnet, HTML form parameters are sent to SAS as macro variables. In fact, web server environment variables and cookies are also sent as macro variables. Consider the following HTML form:

```
<FORM ACTION="report.sas" METHOD="GET">
<SELECT NAME="country">
<OPTION>USA<OPTION>Canada
<OPTION>Mexico
</SELECT> <INPUT TYPE="submit" value="Show
Report">
</FORM>
```

When this form is submitted to the program "report.sas", Onyx creates the following SAS macro variables:

```
GLOBAL _IP 127.0.0.1
GLOBAL _BROWSE Mozilla/4.0 (compatible;
MSIE 6.0b; Windows NT 5.0)
GLOBAL ONYXSESSIONID
369802410416145830052829111801727
GLOBAL COUNTRY Canada
GLOBAL ONYXSESSIONID_ C
GLOBAL _IP_ E
GLOBAL _BROWSE_ E
GLOBAL COUNTRY_ G
```

You will immediately notice that the form parameter named "country" has generated two macro variables, COUNTRY and COUNTRY_. COUNTRY holds the value chosen by the user and COUNTRY_ contains one letter indicating what type of information the macro variable COUNTRY holds. In this case the G indicates that it is form data sent via the GET method (indicated in the FORM tag). The other variables marked as E indicate that they are environment variables (notice the browser type). You can use these macro variables just as you would any other:

```
libname sasdata 'my-data-directory';
ods html file= webout (dynamic);
proc means data=sasdata.sales;
title "Sales summary for country
&country";
where country="&country";
var amount; run;
ods html close;
```

This simple proc means uses the macro variable to create a title and form a where clause. One item to note: using macro variables in this way poses a security risk. Since macro variable references with ampersands are compiled as part of the SAS code at run time, a malicious user could enter macro statements, unbalanced quotes, even their own procs or data steps directly into the variable. To check the contents of a parameter before using it you can use the symget()/symput() functions in a data step. These functions store the macro variables as data step variables which are immune to the same compiler dangers.

USING ODS TO GENERATE ADVANCED HTML OUTPUT

Since the birth of ODS in version 7 of the SAS System, there have been many conference papers, books, and tutorials explaining the details of how to use it. Instead of focusing on ODS this paper will simply present how to use it in conjunction with Onyx or SAS/Intnet. A simple example was presented above. The proc means was enclosed in two ods statements, **ods html file**, and **ods html close**. Use the first statement to begin capturing procedure output and the second to finish capturing it. The keyword dynamic is important. It tells ODS to add a required HTTP header to the output it generates. See the SAS Online Doc for complete documentation on ODS. It includes many options and is a very powerful way to create HTML (and other formats) output.

USING SESSIONS

An inherent problem with web-based applications lies in the fact that they are "stateless". That is, each time a user requests a web page from a web server, the web server does not know it is the same person or "session" making another request. So how do programs on the server know that the same user is requesting a report that has just logged in? The simplest way that web applications track this information is through the use of cookies. Most of us are familiar with the cute term by now because of the issue of security. Web sites actually store information on our own hard drives and can read that information each time we request a page from their sites. Onyx uses only one cookie: the Onyx session ID. This value lets Onyx know which parameters and/or data sets correspond to the current user.

All information gathered during a session is stored in a directory on the server and assigned the libname SESSION. Thus, SAS programs can store data sets or other items in this libname and it will be available each time the user requests pages from the same browser and computer. Additionally, SAS macro variables are saved across requests in the same session if the special prefix **_ONYXSESSION** is used for each variable. Thus you could have one page that checks a username and password and if successful could store a session variable with the username. Any subsequent programs could check to make sure that session variable exists before granting rights to run the program. Sample code:

```
data _null_;
  length username password $50
        where $100;
  username=symget('username');
  password=symget('password');
  where='username='||quote(username)||
        ' and password='||
        quote(password);
  dsid=open(
    'sasdata.users(where=('||
    trim(where)||'))');
  if dsid then do;
    if attrn(dsid,'ANY') then
      call symput(
        '_onyxSessionUsername',
        username);
    rc=close(dsid);
  end;
run;
```

Another program could then check the value of **&_onyxSessionUsername**:

```
%macro report;
  %if %length(&_onyxSessionUsername)=0
  %then %do;
    data _null_;
      file _webout mod;
      put 'Access Denied!';
    run;
  %end;
  %else %do;
    ** REPORT CODE HERE **;
  %end;
%mend;
%report;
```

USING ONYX DYNAMIC HTML

A unique and exciting feature of Onyx is the built-in support for the Onyx dynamic HTML syntax. This syntax lets programmers develop HTML pages with Base/SAS code embedded directly in the HTML. Simply enclose your SAS code in the now familiar **<% %>** tags (similar to Java Server Pages and Active Server Pages). Macro variables can also be resolved directly in HTML! Here is a sample of an ODHTML page:

```
<HTML>
<HEAD>
  <TITLE>My page</TITLE>
</HEAD>
Date/Time program executed:
%sysfunc(datetime(),datetime.)<br>
Date SAS session was started:
&sysdate9<br>

<%
proc sql noprint;
  select nobs into :nobs from
  dictionary.tables where
    libname='DATA' and memname='CONTACTS';
quit;
%>
Observations: &nobs
Proc freq:
<%
ods html file=_webouta;
proc freq data=sasdata.contacts;
  table first last;
run;
ods html close;
%>
</HTML>
```

A major advantage to using this syntax is that developers can use standard web-page editors to build pages. Many commercial and free editors recognize the **<% %>** tags as server-side scripts and simply ignore the contents at design time. ODHTML also makes use of the natural iteration of the data step that allows HTML to be embedded directly into the data step itself. See the Onyx documentation for examples of this powerful feature.

DEBUGGING PROGRAMS WITH ONYX

Special consideration has been taken for debugging programs with Onyx. Simply include the following line of code in your program to return the SAS log to the browser:

```
%put NOTE: ONYX LOG;
```

Or if you would like to see the SAS log only if an error occurs:

```
%put NOTE: ONYX ERROR LOG;
```

After the program runs, the log is scanned for errors. If an error is found and the appropriate debugging directives are

found the log will be returned to the user. This is extremely helpful in testing your programs.

WRITING SCL PROGRAMS

Onyx fully supports the use of the SAS Component Language to develop web-based programming content. SCL is a rich language that allows rapid development of programs and is well suited for this kind of programming. We will take a look at obtaining form parameters via an SCL list, the structure of the Onyx Drone object, and how to use submit blocks effectively to generate HTML output or submit Base/SAS code.

OBTAINING FORM PARAMETERS FROM A LIST

Each time a user makes a request via the PROGRAM request type, SCL lists are generated that contain all parameters sent to Onyx. These values correspond to the macro variables created, in fact, they are used to create the macro variables. The list you can access is stored in the local environment list and is named simply ONYX. SCL code to obtain these values:

```
Dcl char username;
init:
  onyx=getiteml(envlist(),'ONYX');
  username=getnitemc(
    getniteml(onyx,'P'),
    'username');
return;
```

You can see from this example that the ONYX list is made up of sub-lists corresponding to the type of data being sent.

THE DRONE OBJECT

Because Onyx is developed using SCL, it is tightly integrated into the SAS/AF environment by exposing internal methods and properties of the Drone object to user programs. The Drone object is what powers each drone and contains a number of useful methods and properties that your programs can access. Two examples we will look are the methods clientData() and getVarS():

```
Dcl object drone;
Dcl char username password loggedin;
init:
  drone=getnitemo(envlist(),'Drone');
  ** GET SESSION VARIABLE LOGGEDIN **;
  drone.getVarS('loggedin',loggedin);

  ** GET USERNAME AND PASSWORD FROM FORM **;
  drone.clientData(
    'username password',
    username,password);
  drone.clientData('G',
    'username password',
    username,password);
return;
```

Many other methods and properties are available. If you are interested, they are described in the Onyx documentation.

USING SUBMIT BLOCKS EFFECTIVELY

SCL submit blocks are a powerful feature of the SCL language. Most often, they are used to submit base SAS code or SQL code (hence the name "submit" block). But when used with no qualifier, the submit statement actually only places text into what is called the preview buffer. A buffer of text is stored in memory and the preview() function can manipulate that text. Submit blocks can also resolve SCL variables preceded by an ampersand. Here is a simple program that uses submit blocks to write an HTML page:

```
Dcl char date;
init:
  control asis;
  date=putn(datetime(),'datetime. ');
  submit;
  <HTML>
  My web page here - created on
  &date
  </HTML>
endsubmit;
rc=preview('file','_webout','append');
rc=preview('clear');
return;
```

A very important option when using submit blocks is to include the control asis statement (in bold above). This tells the submit blocks to leave the contents formatted as-is. Otherwise, it will attempt to format it optimally for SAS code, which may severely alter the HTML you wish to write. Submit blocks can be used inside of do loops and be executed conditionally, making them a powerful choice for HTML development.

ADVANCED FEATURES

CUSTOMIZING ONYX

Another exciting feature of Onyx is the ability for SAS/AF developers to change existing request types or add new ones. This means that new features can be added to Onyx. When customized, Onyx acts simply to connect the remote user to the SAS System and individual developers can decide how to act on the OSAP request. A number of built-in request types come with Onyx, described previously, which can be enhanced as desired.

USING OTHER CLIENTS THAN THE WEB

As mentioned on the first page of this paper, the use of the Application Dispatcher does not apply just to a CGI script on a web server. Users can telnet to the Dispatcher and enter OSAP requests directly, or other clients can use TCP/IP socket programming just like when reading from web sites. InterNext plans to develop a Java Servlet in the near future that will communicate with Onyx. Other products that could easily communicate with Onyx include Microsoft Office (through the use of Visual Basic), SAS (through the socket access method), or C/C++.

RE-DISTRIBUTING APPLICATIONS

A unique and powerful feature of Onyx is its re-distribution support. Built with the developers in mind, any license holder of Onyx can distribute applications to sites that do not have Onyx licensed. A special set of keys called app keys are generated that encrypt the program names, other program information, Onyx license holder information, and (optionally) the client SAS site ID. Thus you can develop web-based applications in SAS for clients that do not have any web product licensed.

Only SAS catalogs can contain app keys. If your applications makes use of Base/SAS programs (like in .sas files), place them in .source entries in a catalog. When your users request a .sas file from their web server, Onyx will automatically convert the reference into a .source reference and will be able to find your programs.

CUSTOMIZING ERROR MESSAGES

System error messages are taken from HTML files in the Onyx\Java\Errors\ directory. To customize the error messages change those files. AF developers can also add new error files and call them explicitly in their code by using Onyx.Main.Error.Scl. See the documentation for details.

CAPTURING DRONES

Onyx gives programmers the ability to "capture" a drone so that it can only be used by one session. You may want to capture a drone if you want to make sure a user will never have to wait for a session, or to save SCL lists or macro variables in memory or datasets or catalogs in the work directory. Captured drones are not cleaned up in between requests, so libnames, filenames, macro variables, etc. remain undeleted.

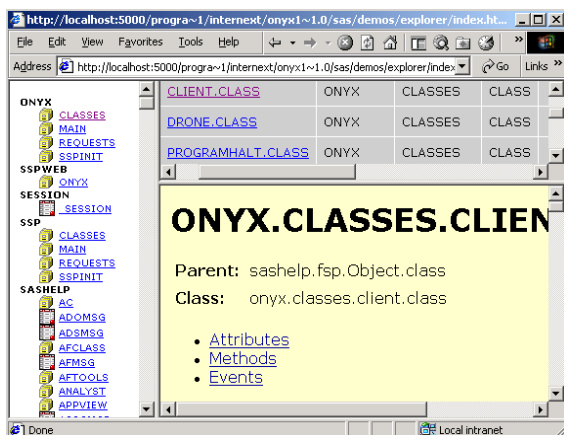
Since you are guaranteed the same session for each request, you can use SCL lists, but keep in mind that the lists may not get deleted properly when the capture is lost so you may consume unnecessary memory. The Application Dispatcher controls who has access to the captured drone. If no request has been sent to a captured drone for an indicated timeout period the capture will be released and a command is sent to the drone to clean itself up. The next time the client tries to use a captured drone they will potentially get another drone so you will need to check to make sure the drone is still captured by the intended user.

To capture a drone simply set the macro variable `_onyxCapture` to "Capture" or "Release" and the specified action will be performed:

```
%let _onyxCapture=Capture; * Capture a
drone;
%let _onyxCapture=Release; * Release a
drone;
```

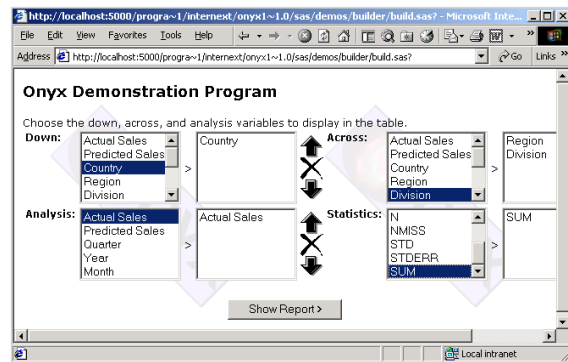
DEMONSTRATION APPLICATIONS

Onyx includes two demonstration applications with source code so that you can test Onyx and learn how to write web-based SAS programs. The first application is the Onyx explorer. This SCL application shows how to use SCL to develop web content. The explorer displays all data sets, catalogs, and catalog entries defined in a Drone. Upon clicking on individual items, users can view details such as proc contents, catalog entry contents, and actual observations from data sets. Here is a screen shot:



The second demonstration program is a simple report builder using data included with the SAS System installation. Users can choose which variables they would like displayed

down the table, across the table, which analysis variables, and what statistics they would like. A proc tabulate is executed with the users choices:



And the output of the proc tabulate:

The screenshot shows the 'SAS Output' window with a table of sales data. The table is structured with Country as the primary row variable, and Region and Division as column variables. The data is summarized by Actual Sales and Sum.

			Region		Division	
			EAST	WEST	CONSUMER	EDUCATI
Country						
CANADA	Actual Sales	Sum	127485.00	119505.00	117984.00	129006
GERMANY	Actual Sales	Sum	124547.00	121451.00	123846.00	122152
U.S.A.	Actual Sales	Sum	118229.00	119120.00	120655.00	116694
All	Actual Sales	Sum	370261.00	360076.00	362485.00	367852

CONCLUSION

Though many organizations are unable to obtain SAS/Intrnet, there are still alternatives for dynamic publishing of SAS information via the web. One such alternative, a fully supported product, is Onyx. This paper has presented an overview of Onyx and examples that take advantage of unique features it has to offer. Whether it is via SAS/Intrnet, Onyx, or some other method, you are encouraged to begin developing web-based content with the SAS System!

OBTAINING ONYX

We will have CDs available at the conference, or you can obtain Onyx software through the InterNext website at <http://www.internext-inc.com>. You can both download the software and obtain an evaluation key that will allow you to develop SAS programs for 30 days. You can contact the author with further questions.



ACKNOWLEDGMENTS

SAS and SAS product names are registered trademarks of the SAS Institute. Other trademarks are the properties of their respective owners.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

David L. Ward

InterNext, Inc.

254 Resnik Ct.

Somerset, NJ 08873

Work Phone: (732) 745-9823

Email (preferred): dward@sashelp.com