**Paper AD06**

# A Hiker's Guide to Web Development

Steve James, Centers for Disease Control and Prevention, Atlanta, Ga.

## ABSTRACT

Building web applications using SAS/IntrNet® Application Dispatcher is a lot like hiking. At first, it is hard. You struggle getting used to your equipment and the new terrain. Then it gets easy as you settle into a pace you like and get into a rhythm. Then it gets hard again as you tire and have to make that final push towards the finish. However, once you get there you can sit back, relax and enjoy having completed the task.

If all that is true, then how should you do it? What can you expect along the way? What kind of equipment do you need and whom should you take with you? This paper is a trail guide for developing web applications with Application Dispatcher and describes the process, pitfalls and practical issues that you can expect to encounter. Specific topics that are covered include system analysis and design, learning Application Dispatcher, coding issues, usability, security, and final deployment.

This paper is designed for users at all skill levels working on all operating systems.

## INTRODUCTION

This paper describes the process of developing a web application and some of the issues that must be addressed. Using the analogy of a hiking trip, the following topics are discussed:

- Planning the Hike
- Hiking Companions.
- Training for the Hike
- "Doing" the Hike
- Finishing the Hike

While explaining some of the benefits of a web application the point of reference for this paper is an environment where IT staff produce ad-hoc reports using The SAS® System. This paper borrows heavily from the author's experience in such an environment and subsequent development of a web application called WISQARS. Several examples in its development are cited. Web development in other environments will be similar to what is described here but may have a slightly different set of benefits and challenges.

## WHY GO IN THE FIRST PLACE?

Why go on a web application development "hike?" Mainly because it's fun. But normally that excuse doesn't satisfy your boss to justify embarking on such a journey or for purchasing a SAS/IntrNet license. What can you hope to gain in this endeavor? Benefits include:

- **More Users have Access to the Data** – Worldwide coverage of the Internet assures more people will have access to data than could be accommodated by ad-hoc requests.
- **Empowers the Data Users** – Users can get the information that they want quickly and accurately. It allows the user to interact with the data without repeated requests to the programmer.
- **Business rules are enforced** - Appropriate logic and analysis procedures can be used to ensure that the correct results are produced.
- **Reduces the number of Ad-Hoc Requests** - Programming staff are relieved of the burden of ad-hoc requests and can focus on more pressing IT needs.
- **Leverage existing SAS Skills**
  Using Application Dispatcher allows you to develop applications without the need to learn an additional language like Java with WebAF™.

**SUCCESS STORY – INJURY STATISTICS FROM CDC**

Prior to 2000, Injury statistics were available from CDC only from ad-hoc requests and a limited number of static web pages. In March 2000, CDC launched a web-application called WISQARS (pronounced "whiskers" for Web-based Injury Statistics Query and Reporting System – www.cdc.gov/ncipc/wisqars). It provides information on fatal and non-fatal injuries, and leading causes of death in the US. Since its development, WISQARS now:

- Handles an average of 1,000 requests per day
- Has over 3,000 references in Google
- Allows automatic generation of reports previously unavailable, such as those with non-standard age-groups (e.g. ages 6-16, 42-43, 55)
- Has dramatically reduced ad-hoc requests.

By virtually any measure, WISQARS has been a huge success because it allows people to get access to the data that they need and want. Using Application Dispatcher CDC has been able to come up with a system that does more with less and that's a success by almost any measure.

## PLANNING THE HIKE

In the "Planning the Hike" stage, also known as system analysis and design, you determine where you are, where you want to go, and what is the best way to get there. This stage is by far the most important but also the one most likely to be shortchanged. The following steps listed here are essential to the success of your system and will be discussed in detail.

- Where am I now?
- Where am I going?
- How will I get there?
- How we really do it.
- Web-application issues
- Other issues
- Summary

The overall goal of the planning phase is to determine all of the requirements of the system up front before actually starting to build it. Studies have shown that problems that cost only $1 to solve in the design phase can cost $1000 to solve in the implementation phase. In the design phase it may be simply changing a number in a document, rather than re-writing lines of code and changing multiple databases. Careful planning helps eliminate those costly fixes.

It should be noted that this is not the only way to do system analysis and design. There are other models to use such as Prototyping, Spiral and Rapid Application Development (RAD). This approach described here most resembles the waterfall method. The main point is that it should be done in a systematic fashion and not have the system just "thrown together."

### WHERE AM I NOW?

The "Where am I now?" question is really a host of important but simple questions. You will want to ask these questions:

- Who are the staff available to work on the project
- What is their skill level?
- What software and hardware do you have for your project?
- What kind of budget do you have?
- What is your staff's experience on projects like this?

Generally, more is better. The larger the staff, the bigger the skill set, and the bigger the budget makes the trip a lot easier than those with small staff, little skill and little or no budget. Making this determination up-front will help you to determine what are reasonable goals for your system and make some prediction of the likelihood of success.

### WHERE AM I GOING?

Like the "Where am I now?" question, the "Where am I going ?" question is where you determine of the specifics of what the system will do. Questions include:

- What will the system do?
- Who are the users we're trying to reach?
- What specific information must the system provide?

**HOW WILL I GET THERE?**

When all of the "Where am I going" and "Where am I now?" questions are completely answered, you can begin to tackle the "How will I get there" question. This is the creative step in the process where you get to figure out how to get from where you are to where you want to be.

Spend a lot of time brainstorming about possible solutions. Think about all the possibilities that you could choose. Do not bias yourself too much at this point. Some of us have tunnel vision in this area and can think only of solutions that deal with SAS and/or web applications. It blinds us to novel and innovative ways of solving problems.

Moreover, do not let your lack of technical expertise influence the design excessively. You may be tempted to omit a feature for your application because you do not know how to implement it. The system may be better served if you allow for that feature but add it later. You don't want to limit your system by what you can do at the moment. Try to differentiate between things that **you** can't do and what **SAS** can't do, and exclude only the later from the design rather than the former.

**HOW WE REALLY DO IT**

If we were to hike the way some of us develop systems, we'd end up starting out ill-prepared and having to come back several times for additional equipment or directions. Even worse, we might find ourselves out too far to go back and end up forging ahead and ending up with a round-about route to our destination that was twice as far and twice as hard as it needed to be. You need to ensure that you have determined all of the requirements of the system before you begin developing a solution for it.

Suppose you need to go from Georgia to Maine within the next twelve months. You decide to hike the Appalachian Trail. After weeks of buying equipment, reading maps, and doing short weekend trips, you are about to start your trip. Then you find out that you omitted one detail: you need to take a grand piano with you! Because you did not find out all of the requirements up front, your weeks of preparations were all wasted.

A real-life example of this occurred during the development of WISQARS. An on-site contractor decided that he wanted to write his own version of one of the reports that WISQARS was going to produce. Moreover, he was quickly able to develop a system that did the report. However, his system used only the most recent three years of data, and to accommodate all the possible requests a user might make, he created a SAS dataset for each possibility (e.g. a dataset for years 1, 2, 3, 1-3, 1-2, and 2-3) for a total of six datasets. He did this to speed response time because the data would already be summarized by how it was reported. However, what he did not take into consideration was that the system needed to be able to report on data from as many as the previous 20 years. To do that his system would have had to create over 200 datasets in the first year, 300 datasets in year 5 and more each year after that. Maintenance of that system would have been a nightmare. Had he more thoroughly determined the system's requirements he would have designed his system differently.

**WEB APPLICATION ISSUES**

For web applications, there are other questions that must be addressed. They include:

- Who and where are the users?
- Minimizing response time
- Accessibility concerns

***Who and Where are the Users***

Whether or not your application is only available within your company, available to only selected outsiders, or available to everyone is important. For one, if affects how many people might be using your system at any one time, which in turn will impact response time (see "Response Time" below).

Secondly, if your application is used by those outside your company, you may not be able to specify specific browsers for those using your application. CDC commonly has users with older versions of browsers (e.g. version 4 of Internet Explorer™ or Netscape™ or earlier). If they cannot access the system, they voice their concern. You might be tempted to think that it is a non-issue because anyone can get the newest version of many browsers for free, but in practice you can't always force people to use a specific browser on your application.

Also, for intranet/extranet applications, you may be tempted to assume a higher degree of familiarity with the data, although this should be discouraged. Usability testing has shown that even experienced public health professionals have trouble with so-called obvious instructions on the CDC website. Alternatively, it may be another staff person doing the query for the intended target audience and be unfamiliar with the terms. In general, you rarely go wrong by trying to underestimate the knowledge of your users.

***Response Time***

In an ad-hoc environment, you likely did not concern yourself with response time. Since the people making the request were likely not standing over your shoulder, whether the system took a few seconds or a few minutes was not very significant. However, in a web application, it matters a lot. Users tend to loose concentration after ten seconds, so it's good to try to make most if not all queries faster than that. Users may be a little more tolerant of delays on the web from other interactive environments since even static pages sometimes can take 20-30 seconds to load. However, response time will likely be a key element in how you design your system.

One way to reduce response time is to pre-summarize the data using a tool like PROC SUMMARY. This type of solution works particularly well if your data are static. At the CDC it's common to receive data files from various organizations on an annual basis. These data rarely change once issued. Once released, you can summarize the data all the possible ways it can be reported, store the results in a SAS dataset and then create indexes to speed retrieval. Your application then simply fetches the already-summarized data and displays it to the user. Other solutions might include separating the data based on how it's retrieved or upgrading the hardware.

***Accessibility***

Section 508 (accessibility) compliant applications are those that provide a "comparable experience" for users who have handicaps and those who do not. One aspect of that is to make sure your application can be used by text readers that essentially dictate the text of your web page to the user. In addition, it is important not to use color as a means of conveying information, since a significant number of people are colorblind. That does not mean you can only use black and white pages, but you wouldn't want your application to have instructions like "For report 1, press the red button, for report 2, press the green button…" Instead, you might say "… press the red button labeled 'Report 1.'"

The impact of this requirement can be quite significant. It may even entail your developing essentially two different versions of your web application, one compliant and one that is not. Your webmaster would be the best person to help with these issues (see discussion on the webmaster in the "Hiking Companions" section below).

**OTHER PLANNING ISSUES**

There are a number of other planning issues that need to be discussed. A partial list and discussion of them follows.

***Documentation***

One important point is that it's good to write down all of these "Where am I going?" questions and answers so that there can be clear communication between you and the other stakeholders of the system. Writing it down helps everybody understand what the final system will do. If you have it written down in a document and all parties agree, then you have objective criteria on which to judge the success or failure of your system. Additionally in writing the documentation, questions will arise that illustrate situations or conditions that you hadn't thought of and need to account for.

***Breadcrumbs***

While it didn't work out so well for Hansel and Gretel, documenting how you get to your final destination is a good idea. Record why you chose a particular path over another at significant points in the process. It will save you the hassle of retracing your steps should you need to revisit a decision. There was the story of the team designing a system for the US Government and came to the point where they felt they needed to go back and determine why a particular path had been taken over the available alternatives. It cost $1 **billion** to determine why they had chosen what they did. One could only imagine how much time, money and effort could have been saved if someone had taken even just a few moments and written down some of the reasons for the decision on a piece of paper.

***Data Dictionary***

One item in this process that can be extremely helpful is a data dictionary. Not only does it provide definitions for terms to those that are not experienced in the subject matter, it also forces some questions that need to be asked while preparing it. Even for a variable as simple as Sex, there are a number of questions to be answered about its length, its type (character or numeric), presence or absence of missing and/or unknown values, etc. It also becomes a useful guide when programming because the data elements attributes are already defined.

***Data Updates***

It is important to consider the impact of the design on adding additional data to your system. For instance, if you update your system annually with new data, you may want to store each year's data separately. When new data are received, you can add them to the system without affecting the other data that's already been loaded and tested. You can test the new data separately. That's a big benefit.

### *Naming Your System*

Another seemingly trivial but in fact important tip is to pick a "cool" name for your system, and pick it as early as possible. Once a name gets associated with a product, it's very hard to change it. With the development of WISQARS, that name was given fairly early, and a logo was even designed for it. It made a difference in how the system was viewed, not only by users but by the people developing it. The converse of that was a group at CDC that was stuck with the name SLUGs (for SAS Lead User Group) and it was never able to get rid of it despite how unpleasant sounding it was.

### *Testing*

Testing a system once it's completed is often a difficult task in that it's hard to know if you've covered all of the bases. A good technique is while developing your system requirements, write test cases for each requirement. For instance, you may have a business rule that defines every number above a certain value as being statistically significant. These numbers should be noted on the report. What you would then do is to have a separate test document with a statement that tells to check to see if numbers are appropriately flagged as significant or not.

### PLANNING THE HIKE SUMMARY

Addressing these problems in a requirements document is a lot easier than in SAS code. Remember that all of these questions have to be answered at some point anyway so not writing these documents does not save you as much work as you might think. At some point, for instance, you're going to have to decide how the variable SEX is defined (numeric/character, length, values, etc.). Answer these questions in the design phase so that when the changes come, as they invariably do, they can be incorporated without significantly affecting the system.

Clearly a lot more can be said about system analysis and design than is said here. But remembering and answering the simple questions: "Where am I now?" "Where am I going?" and "How will I get there?" will help ensure that you build a successful system.

## HIKING COMPANIONS

Just as it is safer to go hiking with at least one other person, developing web applications is usually a team effort. In addition to the developer role, there are two additional roles that are needed in web development. They are the Webmaster and Network Administrator. Your company may have several people performing each of these roles, and may have more than one job titles that fulfill it a given function. But for the purpose of this document, each role will be written as if it were a single individual and a single job title.

### WEBMASTER

The webmaster is that person in your organization who is responsible for your company's web site. She will help you integrate the application with the general look and feel of the company's web site and inform you of other company policies such as requirements for certain browsers and compliance with Section 508 (accessibility) requirements.

A webmaster will also play a role in making your web pages more usable. Web page design is a lot more than making a web page look "pretty." It's designing web pages with the awareness of human behavior. We've all experienced web sites that are poorly designed and difficult to use. The job of the webmaster is to identify and eliminate any difficulty your pages might present.

Webmasters will likely be the first person to look at your system from a user's perspective. She will not have been immersed in the subject matter as you have and can look at it with "fresh eyes." This will allow her to make some very useful comments about how easy your application is to use. Also your webmaster may suggest usability testing to find hidden problems in your application. Usability testing involves watching users use the system, noting areas of difficulty and confusion. At CDC usability testing has proven to be very insightful in determining what is and what isn't obvious to our users; and there have been many surprises.

It's always a good idea to get the webmaster involved as early in the development of the project as possible, even if you're in the requirements gathering phase. If nothing else, it shows courtesy to the webmaster and indicates that you value her input (which you do even if you don't know you do). Additionally she may help you with company policies and/or clearance procedures that can affect what you do or how you do it. Getting these issues resolved early in the process may save you time later down the road.

### NETWORK ADMINISTRATOR

The network administrator is the other person who needs to be involved in the development process. This is the person who installs and maintains SAS/IntrNet on your server, provides access to enterprise data, handles network and firewall security, and other network issues.

Before you can begin to use SAS/IntrNet you must interact with your network administrator because he determines some of the parameters that you need to get your application to work. In addition to general networking issues, the network administrator needs to give you at least these 4 pieces of information:

1. Where do you store your HTML and what is the URL to use to refer to it?
2. Where do you store your SAS data and programs? How do you refer to them?
3. What is the name of the SAS/IntrNet "service" to use?
4. Where is the SAS Broker to run SAS/IntrNet and how do you refer to it?

These are minimum requirements that you will need to begin developing your application. Additionally you will want to talk with your network administrator about test and production environments and the number of people you expect to use your system. There are components in SAS/IntrNet that the network administrator can use (i.e. Pool Services and Load Manager) to help make things run smoothly, depending on the needs of your application.

Like the webmaster, you'll want to get the network administrator involved in the project during the planning stages to help with issues of security, networks, and other corporate policies.

## TRAINING FOR THE HIKE

Now that you've identified where you are, where you're trying to go, and whom you're going to take with you, you need to get and learn how to use the equipment that you'll need. Here is a list of some equipment that you might consider using on your trip:

- A good text editor
- HTML editor and basic HTML knowledge
- JavaScript References and/or training
- SAS/IntrNet training

### TEXT EDITOR

Surprisingly, you might not use the SAS Enhanced Editor for the bulk of your development work. If your HTML and SAS programs reside on a server that is not directly connected with your PC, you may need to use FTP to pass these files back and forth. Even without needing FTP capability you may choose to use an editor different from the one SAS supplies.

UltraEdit (www.ultraedit.com) is a very nice and very powerful editor that is the preferred choice of many SAS users for editing text files. Among many other features it offers syntax highlighting like the Enhanced Editor. Of particular importance to some is that it allows you to edit files on remote server seamlessly. It cost approximately $35 US.

### HTML EDITOR AND BASIC HTML EXPERIENCE

An HTML Editor such as FrontPage® or Dreamweaver® can come in handy when developing the HTML forms that your application may use. It's also a good HTML reference because you can design what you're trying to do in the editor and then look at the HTML code that it generates to see how to do it.

Be sure to consult with your webmaster before proceeding. FrontPage is one editor that may be incompatible with your website, and in some cases can cause existing pages to stop working. However it's has the same flavor of other Microsoft products and may be easier to use than Dreamweaver. Professional web page designers and webmasters often use Dreamweaver. Also free HTML editors and HTML tutorials are available on the Internet.

### JAVASCRIPT

One issue that you tend not to worry about in the ad-hoc environment is input errors, since a programmer is doing the input. That's not true of the Internet. Here you must ensure that the user does not input invalid selections.

JavaScript is a language that you insert in your HTML code to extend the capability of HTML. It's particularly useful for data validation to ensure that the user does not put in invalid data in a field or put inconsistent data across multiple fields (e.g. Sex of 'Male' and 'Is Pregnant" = Yes).

A good technique to use whenever possible is to have the user make selections with drop-down menus rather than textboxes. This ensures that the user doesn't inadvertently input an invalid entry, and the user can see what choices are allowed.

The advantage of JavaScript is that your browser likely supports it and there's nothing to buy. It also ensures that the data validation part is done at the client's machine rather than the server. This speeds up the process a lot for the user and makes it easier for the developer. Data validation at the server level is slower and can be tedious to code for in that you have to generate your own error messages in HTML using SAS code. There are many Internet sites where JavaScript examples and tutorials are available at no charge so it's not too difficult to learn. Similarly you could also use Visual Basic and ASP pages to do the same thing.

### SAS/INTRNET TRAINING

Since this paper assumes that you'll be creating a web application using Application Dispatcher, you certainly would need to know how to use it. There are several approaches, not all of which are mutually exclusive. You can learn on your own, study examples provided by SAS or others, or take a training class from SAS or another vendor.

Application Dispatcher is difficult at first. It can be confusing keeping track of where each file needs to be stored and how to refer to it and it's one of the biggest challenge facing the first-time developer. But once those things are clear in your mind, the main challenge becomes the SAS code itself. That's a major strength of Application Dispatcher is that you can use your existing SAS skills and create web applications. You do not have to learn a new language.

## "DOING" THE HIKE

Now comes the fun part. All you have to do is to assemble the tools, get your map and off you go.

### FINALLY YOU'RE HIKING

Once you finally start coding, it truly is the easy part for many since it just involves writing SAS code. Often the focus is to simply get the application to produce a report without concerns about input errors, titles and footnotes and the like. While you must consider those issues prior to deployment, (see "Finishing the Hike" below), you may not code for them at this time. You just want to get something working.

After the response time issues, the biggest challenge that you have in going from an ad-hoc environment to a web application will likely be making your code flexible enough to meet your reporting options. For instance, you may have the requirement that the application produce overall reports, reports by sex or race, and reports by sex and race. To use SAS terminology, you need to report with zero, one, or two "by" variables. Depending on how you produce the report and your skill as a programmer, this is an easy or a very difficult task. One option is to have a section of code for each type. You'd have one section for zero "by" variables, a second one for one variable and a third section for two "by" variables. This accomplishes the task in a straightforward manner but there are three different places to change the report. You still will likely need to use a macro to ensure the correct section executes, however.

Another choice is to use macros to add the flexibility to the code for the report. It has a different maintenance cost, however. What it saves in compactness it adds in complexity. Your skill and personal style as a programmer may dictate how you resolve that tension. However, do not forget that at some point there will be someone else who has to understand the code you wrote (perhaps even yourself), so simpler is often better.

### LOSING THE TRAIL

Debugging with SAS/IntrNet is not as easy as running code interactively in Display Manager. In addition to the normal types of SAS errors that you can get you also may have problems with SAS/IntrNet itself. Usually these are caused by misspelling or something similar on your part. This is where it helps to be on good terms with the network administrator because he can help with identifying the culprit for these types of problems.

However, there are some steps that you can take to help minimize the effect of SAS errors in your code. The first is to write and test blocks of code in Display Manager prior to incorporating it into your web application. It's easier to debug there and it's a good place to see if the code works. If it works in Display Manager and not with SAS/IntrNet you've narrowed the problem to SAS/IntrNet.

Secondly it's a good idea to put a drop-down box on your HTML Form that allows you to specify the _DEBUG option you want for each run on the form. The value of _DEBUG determines what error-checking logs get displayed. A value of 128 shows you the SAS log while 131 shows you the SAS log as well as the parameters passed to SAS from your HTML form. A value of 0 shows neither of these. Rather than hardcode it into a hidden field as you'll do later on, add a drop-down box so you can easily change it to see the log. Depending on how common your errors are, you may want to vary the default value of _DEBUG by moving the HTML word "selected" to the desired place in the code below:

```
<P>Debug Option <SELECT size=1 name=_debug>
 <OPTION selected>0</OPTION>
<OPTION>128</OPTION>
 <OPTION>131</OPTION>
</SELECT> </P>
```

This is particularly useful after you're at the point where you're not changing the code as often and you may want to suppress the log most of the time and only look at it on the rare times you have problems after changing something. You'll remove it before your application goes production.

Finally, it's not always easy taking code you've written for SAS/IntrNet and trying it out in another environment such as Display Manager. You end up needing to specify in a %LET statement the parameters that would normally come from your form. You'll also need to change all of the references to _webout on your ODS or FILE statements. You also may have to change any file paths that may be different.

If you really are stuck, a good technique is to write your work files to a permanent library so you can see what they actually contain. Additionally the MFILE option, which writes your resolved macro code to a file, may be helpful as well.

## FINISHING THE HIKE

Now the hard part. You've been writing a lot of SAS code taking minimal effort concerning issues such as data validation, input errors, help files, data correctness, and security. However, each of these areas must be addressed before you can finish and deploy your application. And they may take longer than the coding you've done up until this point. They include:

- Testing
- Security
- Help documentation
- Common look and feel
- Print version of report
- Output formats

### TESTING

Testing a web application is critically important, especially for Internet applications. The reputation of your organization depends on it. Combine that with the fact that you will not know whom to notify that they got wrong information and the importance is even more obvious.

However, **many programmers do not do sufficient testing**. They assume that the algorithm that they used will solve the problem. Additionally they assume that their code correctly implements the algorithm. They can be wrong on both counts. A good rule of thumb is to assume that the program does **not** work and then prove to yourself that it does.

Testing a web application is different from testing the results of an ad-hoc report. For one, you may need to test the data themselves. If you've summarized the data for efficiency's sake you have to verify that you summarized them correctly. One technique is to get someone else to create the data using the same specifications you used, perhaps using a different method and see if the same results are produced. This is a good way to find errors in both the method as well as the actual coding. In addition, if there are published reports or other sources of verifying the data they should be used to verify your results.

Another issue is the need to test the HTML form itself. You must ensure that for every selection that the user can make, the appropriate action is taken. For instance, if your form allows the user to select any one of the 50 US States, you would have HTML code like this:

<input type=checkbox name=state value=GA> Georgia

You need to ensure that the value of that checkbox (e.g. "GA") and the text beside it (e.g. "Georgia") match. It is very easy to have a mistake hidden in your HTML form, particularly if you have many possible entries.

One test that's easy to forget is to test your application on different browsers. Your application will likely act at least slightly different between browsers. Minimally you should test on the current versions of Internet Explorer® and Netscape®.

You want to be as thorough as possible in testing the application prior to deployment. As some have said, "All systems are tested for errors. Some prior to deployment, others afterwards, but all systems are tested. " Aside from the obvious benefits, it helps avoid that sinking feeling in the pit of your stomach when someone calls with a question about your application. If you've tested it thoroughly you can feel confident that your application is correct. Thorough testing before deployment can significantly reduce your consumption of antacids.

### SECURITY

If you're web application is for internal use only, then security may not be a big concern. However, if you're releasing your application on the Internet there are some things to be concerned about.

The first thing is to change the METHOD=GET to METHOD=POST on the <form> statement in your HTML. The main difference between the two is that with GET, your parameters display in the address line of your browser. This is handy for debugging purposes and allows generated reports to be bookmarked. However, it shows important details about your application to others and thus is a security risk.

Secondly, your want to protect yourself from “dirty” input data: characters such as quote marks that could cause your application to malfunction. There are two methods for this. One is using the UNSAFE= option on PROC APPSRV which the network administrator would control. The other is by using %SUPERQ macro function and the SYMGET call routine in the DATA step. Both of these techniques are designed to nullify the effect of any bad data. For instance, if you had a text box where a person could type in a title for a report, that’s a place where the wrong character can cause problems. If your SAS code looked like this:

```
Title “&mytitle” ;
```

and the user types in:’ Report” of’ (i.e. embedded an unpaired quote mark), SAS would resolve the title statement to the following:

```
Title “Report” of“ ” ;
```

and a syntax error would be generated. You want to avoid this from happening whether its from malicious or benign intent. To avoid the above problem, you would code it as this:

```
Title “%SUPERQ(mytitle)” ;
```

Your network administrator can tell you what characters are used for the UNSAFE= option for PROC APPSRV and help you with other security issues.

**HELP DOCUMENTATION**

One of the most onerous tasks in this whole endeavor is writing the help file, particularly if your application is going to be used on the Internet. You pretty much have to spell out in detail everything that there is to know about the application including the source of the data, how the numbers are calculated, when’s new data going to be added, etc. **Do not underestimate this effort.** It can be quite significant if you want to fully document the system. For the WISQARS application, it took an estimated 2-3 months of full-time work to complete. Your webmaster will be able to give you some guidance on the types of things to include in your help file.

There are several reasons why the help file is useful despite the amount of effort it takes to write it. The first is that it’s one more check to ensure that you have accounted for all possibilities that your system should handle. While you are thinking about some of the questions that users may want answers to it may occur to you that you have not adequately accounted for every possibility. It also may lead to better footnotes on the reports. It may remind you of missing links between a term on a web page and its definition in the help file. In addition, it may force you to learn things about the data that you never took the time to find out.

Additionally a hidden benefit in developing the help file is that it helps the IT staff who must maintain the application. It documents all the business rules and data anomalies that apply to the system. It is also the first logical choice for answers to questions about the system. Certainly, it cannot be the only documentation for the system, but it is a good start. It can be a big benefit to those who will do maintenance on the system or work on the data in another context.

It should be noted that there is some reason to believe that help file is not read by many people, making its creation hard to get motivated for. However there are still the benefits noted above to consider, though one may consider scaling back the effort from what you might originally think is necessary.

**COMMON LOOK AND FEEL**

The web has matured to the point that it’s no longer acceptable to have pages with a variety of formats within a given site. Users have come to expect a common look and feel within a company’s website. However since the pages are created dynamically the application itself has to provide that look and feel. Fortunately, there are a number of easy ways that you can have all your web pages with a common appearance. Perhaps the best way is with a DATA step within the application. In its simplest form, code to give your output a standard header would use code like this:

```
data _null_ ;
  infile “company_header.html” ;
  input @80 text $80. ;
  file _webout ;
  put text $80. ;
  run ;
```

You could follow this code with whatever other output mechanism you need, such as PROC REPORT or another DATA step. A similar set of code could be used for the footer as well. The advantage of doing it this way is that you as the developer control it and it’s easy to maintain. The webmaster can create the header and footer HTML files for you and they are separate from the SAS code so it’s easier to maintain.

If your output can vary significantly in terms of width, you may want to eliminate some objects such as side navigation bars to have your report format nicely. The webmaster can help you in making those decisions and providing you with the file to use for your headers and footers.

For a more thorough discussion of this subject you may want to read a paper by this author called *Adding A Common Look and Feel to Web Applications Easily, SUGI 26 Proceedings, and available at:* http://www2.sas.com/proceedings/sugi26/p076-26.pdf . ODS Markup has since added capabilities for this not covered in the paper. For more information, go to http://support.sas.com/rnd/base/topics/odsmarkup/.

**PRINT VERSION OF REPORT**

Users have also come to expect a separate output format for printing from the normal display format.. The print format may be as simple as not producing the standard headers and footers described in the common look and feel section above. The same issues of maintenance and complexity as described in Section III, "Doing" the hike apply. You may want to sacrifice complexity by having multiple places to make changes or vice-versa.

**OUTPUT FORMATS**

ODS has made the ability to provide output in different formats relatively easy. You can output your reports with HTML, PDF, RTF, and Microsoft® Excel® formats. Information on producing output in HTML, PDF, RTF and Excel can be found at http://support.sas.com/rnd/base/topics/templateFAQ/Template_special.html#S18A . Additionally you can provide the data in a comma-separated value (CSV) format for those who do not have Excel and/or want to read the data using other software. More information on creating CSV files can be found at http://support.sas.com/techsup/unotes/SN/009/009460.html .

## CONCLUSION

Web development using Application Dispatcher is a rewarding alternative to the generation of ad-hoc requests. The users get access to more data in more ways with faster results. And the IT staff exchange the tedium of ad-hoc requests for the new challenge of maintaining a web application. There are a number of things to consider and it's not necessarily a simple task, but if done properly, the impact can be huge.


## ACKNOWLEDGEMENTS

I would like to thank Chevell Parker and David Shinn of SAS Institute for their technical assistance in this paper.

I would also like to thank Maggie Reilly, Colleen Jones, Dionne White, Bob Thomas, Sharon Clanton and Van Munn of CDC for their help in preparing this paper.


## CONTACT INFORMATION

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


Steve James
Centers for Disease Control and Prevention
1600 Clifton Road NE, MS-E57
Atlanta, Ga. 30333
(404) 639-6041 (voice)
(404) 639-8555 (fax)
sjames@cdc.gov