Virtualizing Software Development Life Cycle Environments in SAS®
Robert L. Jones, PJM Interconnection, Norristown, PA

ABSTRACT
Operating system environment variables provide a simple way to embed environment-dependent information (host machines, pathnames to files and folders, 3rd-party data technologies, etc.) into SAS source code. This paper considers how to utilize environment variables using Citrix MetaFrame XP Presentation Server and SAS Integration Technologies as client/server technologies to consolidate multiple SAS server environments onto a single physical machine, thereby saving money in SAS licensing fees.

INTRODUCTION
The software development life cycle (SDLC) is a framework for developing and testing software before implementing production-quality versions thereof. A general approach toward facilitating an SDLC is to create isolated server environments for different SDLC phases of work. In the simplest case, three such environments would exist: development, test, and production. Migrating user-developed applications across each of these environments leads us to ponder how to accommodate instances in source code where the location of program files or data depends on the SDLC environment.

One implementation is to assign each SDLC environment its own physically independent machine. Here, the underlying assumption is that the development, test, and production machines are identical except for their respective DNS hostnames. File and folder pathnames, for example, would be the same in this scenario regardless of environment. This has the benefit that an environment in which a SAS program is experiencing technical issues can be troubleshooted by running it in a mirror environment and making pairwise comparisons between the original environment and each mirror environment. For software license models like that used by SAS, where the cost is a function of the total number of CPUs across all environments, this approach can be rather expensive.

A cheaper architecture is to virtualize two or more SDLC environments on the same machine but use different paths to create logically separate storage areas for program and data files. For example, the development and test environments would reside on one machine, whereas the production environment would reside on a machine without other SDLC environments.

This paper describes how to use operating system (OS) environment variables for virtualizing SDLC environment on a single machine. It first gives a brief tutorial on environment variables for three popular OS families: UNIX, Linux, and Windows. SAS environment variables are also covered but in even less detail. It then lists the environment variables that the author created for environment virtualization as well as the philosophy behind the choice thereof. Finally, it covers two client/server technologies for remote access to virtualized SAS environments: Citrix and SAS Integration Technologies.

All examples presented were on Windows 2000 Advanced Server machines in generalized to UNIX and Linux where appropriate. Also, for brevity, Windows henceforth shall be synonymous with Windows NT, 2000, XP, 2003, and variants thereof.

OVERVIEW OF ENVIRONMENT VARIABLES
DEFINITION
Environment variables, as defined by Wikipedia, are a set of dynamic values that can affect the way running processes will behave on a computer. They are similar to SAS macro variables in that they associate strings of various lengths with symbolic names. However, because their names and values are stored at the OS level, their values can be passed into programs written in languages besides SAS. As such, they provide an administratively convenient alternative to hard-coding program-independent but environment-sensitive information—the names of files, paths, users, and even the host machine—in potentially scores of SAS and non-SAS programs.

Environment variables come in one of two flavors: system variables or user variables. System variables are environment variables that are common to all user accounts. Those built into the OS have default values that are set during installation. Some common UNIX/Linux system variables are:

- HOME: the full path to the user's home folders
• **PATH**: a colon-separated list of folders that the OS shell searches for commands

Those in Windows include:

• **PROGRAMFILES**: the full path to the “Program Files” folder
• **USERPROFILE**: the full path to the current user's profile folder

User variables, in contrast, are defined by users and programs. Unlike system variables, the availability of which is the same regardless of user, the existence and therefore the values of particular user variables depends on the actively running user account.

**SYNTAX**

How an environment variable appears in a script or command-line interface depends on both its context as well as the shell environment in which the environment variable is referenced. Consider, for example, a fictional environment variable named FOO. For simplicity, we will assume the shell for UNIX and Linux is BASH and for Windows is CMD.EXE. One way to display the variable’s value is using the ECHO command, which is common to all three OS families.

- **UNIX/Linux**: `echo $FOO`
- **UNIX/Linux**: `echo ${FOO}`
- **Windows**: `echo %FOO%

In this case, the shell determines the special symbol and its placement before or around (or both if more than one symbol is used) the environment variable name. This is in contrast to setting an environment variable’s value to, say, “Hello, world”.

- **UNIX/Linux**: `FOO=Hello world`
- **Windows**: `set FOO=Hello world`

Here, no special symbols are used; only the command itself depends on the shell. Putting the previous two set of examples together, suppose that an environment variable FOO2 is to be (re)assigned in terms of another, such as when defining the path to a subfolder named “subfoo”, the parent folder of which is represented by FOO.

- **UNIX/Linux**: `FOO2=$FOO/subfoo`
- **UNIX/Linux**: `FOO2=${FOO}/subfoo`
- **Windows**: `set FOO2=%FOO%\subfoo`

We will utilize this technique more in the next section.

As a side note, the names of the environment variables seen thus far have been capitalized. Although not an enforced syntax requirement, it is nonetheless a de facto convention that is observed in UNIX/Linux and Windows. We, too, shall utilize this capitalization convention when naming environment variables.

**ENVIRONMENT VARIABLES IN SAS**

**READING VALUES**

In the previous section, we saw three different syntax conventions for reading environment variable values. To accommodate a potential plethora of conventions, including those from the shell environments of SAS-supported OSes we have not discussed (i.e., z/OS and OpenVMS Alpha), SAS maintains its own environment variable conventions. The first of these is for referencing external files. For example, suppose the value of an environment variable FOO_FILE corresponds to an external data file on a Windows machine.

`set FOO_FILE=C:\data\subdir\example_data.dat`

When a SAS program invokes that environment variable, no special characters appear with it.
data foo_data;
  infile FOO_FILE;
  input col1 $ col2;
run;

The other environment variable convention that SAS follows is for referencing the full path of folders. Specifically, environment variable names are prefixed with an exclamation mark (!). Situations where this convention is useful include reading SAS programs using the %INCLUDE statement and defining libraries of SAS data sets using the LIBNAME statement. An Example demonstrating both situations using the environment variables FOO_DATA and FOO_CODE is shown below.

libname foo_lib1 "!FOO_DATA\subdir\example_lib1";
libname foo_lib2 "!FOO_DATA\subdir\example_lib2";
%include "!FOO_CODE\subdir\example_code1.sas";
%include "!FOO_CODE\subdir\example_code2.sas";

It should be noted that environment variables representing folders when referenced in SAS must appear as the root folder, not as a child folder below the root. This is not so for such environment variables when used in shell scripts or batch files. The next example illustrates these legal syntax distinctions.

- **SAS (illegal)**: libname foo_lib "rootdir\!FOO_DATA\subdir\example_lib1";
- **SAS (legal)**: libname foo_lib "!FOO_DATA\subdir\example_lib1";
- **SAS (illegal)**: %include "rootdir\!FOO_CODE\subdir\example_code1.sas";
- **SAS (legal)**: %include "!FOO_CODE\subdir\example_code1.sas";
- **UNIX/Linux (legal)**: FOO2=/rootdir/%FOO%
- **Windows (legal)**: set FOO2=C:\rootdir\%FOO%

For our discussion, we will only use OS environment variables to represent the full path of important folders.

**SETTING VALUES**

Up until now, the context for setting values for environment variables has been to do so within the shell environment of the OS, whether SAS is running in the background or not. We will refer to these simply as OS environment variables. SAS, too, can be used to automatically or manually set environment variables. Some are built into SAS and are global for all SAS sessions, such as:

- **SASCFG**: the full path to the SAS System configuration folder
- **SASROOT**: the full path to the folder in which SAS is installed

Their default values are set in the SAS configuration file but can be overridden by editing the SAS configuration file (not recommended), from the SAS invocation command line using the SET system option, or from the shell environment of the OS (e.g. Windows). Others are created and set by SAS users on an as-needed basis using the SET system option at SAS invocation as described above or in individual SAS programs using the OPTIONS command.

The built-in and user-defined environment variables set in SAS are collectively called SAS environment variables. Unlike OS environment variables, the lifetime of SAS environment variables is the duration of the SAS session in which they are set. They are mentioned here because of their noteworthiness, not because they are used in the approach discussed in this paper. The author recommends using OS environment variables for passing general environment information to both SAS and non-SAS programs.
ENVIRONMENT VIRTUALIZATION

FILE SYSTEM ARCHITECTURE

Environment variables facilitate environment virtualization for applications by dynamically passing environment-specific information to those applications at runtime. In the case of SAS, the pertinent information is the location of files that are read or written by SAS programs. This includes:

- **Application source code files**: SAS programs and 3rd-party scripts/programs)
- **I/O files**: data (SAS data sets, flat files, spreadsheets, etc.), logs, and reports (HTML, PDF, RTF, etc.)

As such, a discussion of the file system architecture naturally leads to a discussion of environment variable creation.

A simple SAS file system architecture to consider that leads itself to a SDLC is as follows. First, all non-temporary files (source code, data, etc.) are maintained in a folder tree under a single root folder per environment. Label the environment variable representing this folder as ENVROOT. Next, all subfolders underneath ENVROOT correspond to key file categories of interest. Suppose for the sake of argument that they correspond to source code, data, logs, and reports. Label their respective environment variables APPSROOT, DATAROOT, LOGSROOT, and RPTSROOT. Thus, a Windows implementation of these environment variables for a SDLC consisting of three environments (development, test, and production):

```plaintext
set APPSROOT=%ENVROOT%\apps
set DATAROOT=%ENVROOT%\data
set LOGSROOT=%ENVROOT%\logs
set RPTSROOT=%ENVROOT%\rpts
```

where:

- **Development**: set ENVROOT=t:\SDLC\dev
- **Test**: set ENVROOT=t:\SDLC\tst
- **Production**: set ENVROOT=t:\SDLC\prd

A sample SAS program that demonstrates how to reference these environment variables is given below.

```plaintext
filename foo_rpt "!RPTSROOT\dept2\app3\example3.lst";
filename foo_log "!LOGSROOT\dept2\app3\example3.log";

proc printto
   log=foo_log
   print=foo_rpt
;
run;

libname foo_lib1 "!DATAROOT\dept2\app3\example_lib3a";
libname foo_lib2 "!DATAROOT\dept2\app3\example_lib3b";
%include "!APPSROOT\dept2\app3\example_code3a.sas"
%include "!APPSROOT\dept2\app3\example_code3b.sas";
```

CITRIX

The examples in the previous subsection parameterize environment virtualization in terms of a single environment variable: ENVROOT. Said a different way, the value to which ENVROOT gets set defines the virtualized environment. In the architecture this paper considers, two SDLC environments (development and test) are hosted on the same remote machine and the third (production) is on a different remote machine. Therefore, client requests to access the SAS server within the context of each virtualized SDLC environment need to set ENVROOT prior to invoking a SAS session.
One vendor with a client/server technology capable of fulfilling this architectural requirement is Citrix. An older software product of theirs, Citrix MetaFrame XP (now replaced by Citrix Presentation Server 4), runs on top of Windows Terminal Services and allows users to run server-side executables from a client-side graphical user interface (GUI) called the Citrix Program Neighborhood (see Figure 1). The Citrix server publishes these executables to Program Neighborhood as application sets, and Citrix Program Neighborhood represents the executables as icons. Ordinarily for users who access SAS through Citrix, the executable is simply the SAS invocation command (SAS.EXE), followed by an optional set of hard-coded SAS system options. In this simple scenario, clicking on a Citrix Program Neighborhood application icon representing the SAS environment of interest launches the SAS Display Manager (see Figure 2). For more complex scenarios where a sequence of actions needs to precede the SAS invocation, such as in environment virtualization, the server-side executable can be a script that encapsulates these actions together with the SAS invocation command. A simple Windows implementation of such a script could look like the example below.

```
set ENVROOT=t:\sasDev
set APPSROOT=%ENVROOT%\apps
set DATAROOT=%ENVROOT%\data
set LOGSROOT=%ENVROOT%\logs
set RPTSROOT=%ENVROOT%\rpts

"%sasroot%\sas.exe" -awstitle "SAS 9.1 - %sasEnv%"
```

Using Citrix software does have caveats, however. First and foremost, both the client and server machines must be running Windows. Second, Citrix is licensed based on the number of concurrent Citrix connections to the server, not the number of concurrent submitted SAS jobs. Users who have the SAS Display Manager open but are not running SAS jobs in either interactive or batch mode use up license seats and consume server resources such as CPU and memory to maintain their Citrix connections.

![Citrix Program Neighborhood - Corporate Applications](image)

Figure 1. SAS SDLC Environments within Citrix Program Neighborhood (Version 7.100.x)
SAS INTEGRATION TECHNOLOGIES

A more flexible, cheap, and efficient client/server architecture is SAS Integration Technologies. It is especially ideal for facilitating remote access to SAS environments with heterogeneous OS requirements. At its heart is a daemon (i.e., background process or service) called the SAS object spawner, which is installed and operational on each machine intended for SAS use. The object spawner primarily listens for SAS invocation requests from Integrated Object Model (IOM) clients such as SAS Enterprise Guide® and runs specific executables based on the information contained in those requests. The requests contain information such as the machine hostname and port number, which determines the executable that is run.

Figure 2. SAS Display Manager in the SAS Production SDLC Environment

Figure 3. SAS SDLC Environments within SAS Enterprise Guide 3.0
Which executables get run is determined by the object spawner configuration file. The file itself can be in one of two formats: the LDAP Data Interchange Format (LDIF) or XML. In this paper, we will focus on the LDIF version. (The author reserves the right to publish a second paper if SDLC environment virtualization using an object spawner XML file proves doable.) In the LDIF file, executables are specified in the SAS object server definition (see below), along with the machine hostname, port number, and other required and optional information.

```
# SAS Object Server Definition - Foo
#
dn: sasServercn=testServer
objectClass: sasServer
sasServercn: testServer
sasDomainName: foo_domain
sasMachineDNSName: host.foo.com
sasPort: 5307
sasProtocol: bridge
sasCommand: "C:\Program Files\SAS\SAS 9.1\sas.exe"
description: SAS Integration Technologies Server – Foo
```

As with Citrix, the executables that the object spawner runs need not be limited to instances of the SAS invocation command. Much to the contrary, they can also include SDLC environment virtualization scripts such as the following one written for Windows 2000 Advanced Server.

```
set ENVROOT=t:\sasDev
set APPSROOT=%ENVROOT%\apps
set DATAROOT=%ENVROOT%\data
set LOGSROOT=%ENVROOT%\logs
set RPTSROOT=%ENVROOT%\rpts

"%sasroot%\sas.exe" %*
```

It is similar to the one used in the Citrix example, except that it includes a trailing %*. It corresponds to all command-line parameters and purposefully appears at the end of the SAS invocation command line. Without it, running the script would not result in invoking a SAS session.

**CONCLUSION**

Environment virtualization is a cheap alternative to dedicating server hardware for each SDLC environment. The cost differential can be measured in terms of comparative hardware, software, and maintenance expenditures. For multiprocessor machines under the SAS license model, the cost savings by using this technique and physically consolidating, say, separate development and test machines into a single machine with virtualized SDLC environments can be significant.

Environment variables and appropriate file system and client/server architectures are what make SDLC environment virtualization on the cheap possible. Recommendations for successful implementations include the following steps:

1. Plan the file system architecture.
2. Determine which folders are most important, and re-express the paths to them in terms of environment variables.
3. Select a client/server architecture based on considerations such as security, number of users, and server-side OS.
4. Enroll SAS users and management for their buy-in.

Lastly, obfuscate the technical details that make SDLC environment virtualization possible, details that frankly are irrelevant and maybe even confusing to SAS users. Otherwise, us folks who support SAS on the backend risk hearing questions—or worse yet, complaints—from the same SAS users we know and love.
REFERENCES

ACKNOWLEDGMENTS
Thanks to Garrett Allred from SAS Technical Support [SAS us6171596] for his invaluable assistance as the ideas presented in this paper for configuring the SAS 9.1.3 object spawner were first being prototyped. Thanks also to my colleagues Heather Forsythe, Richard Lanning, and Michael Radel, and Fabian Robinson for supporting and encouraging me to evangelize the merits of my team’s accomplishment in my first published paper.

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

Robert L. Jones
PJM Interconnection
955 Jefferson Avenue
Norristown, PA 19403
Work Phone: 610-666-8954
Fax: 610-666-2299
E-mail: jonesr@pjm.com, bob@bobweb.org
Web: www.pjm.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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