Paper 004-2008

Return of the Codes: SAS'®, Windows'®, and Yours

Mark Tabladillo, Ph.D., MarkTab Consulting, Atlanta, GA
Associate Faculty, University of Phoenix

ABSTRACT

Robust applications engage in the give-and-take discussion between commands and return codes. This presentation encourages applications developers to implement comprehensive return code processing. This paper presents three distinct categories. First, we consider return codes from SAS®. Second, we consider return codes from the Windows® (as an example operating system). Third, we discuss development principles for proactively writing your own return messages. The examples draw from SAS/AF® and Windows®, and affect all SAS applications development (including robust SAS Macro development). This paper introduces the rich conversation an application can and should have with its environment. Special attention focuses on error messages and recovering gracefully from unexpected or unintentional results.

INTRODUCTION

In the generic case, an application allows multiple users to retrieve and update information synchronously or asynchronously from other users and a data warehouse. The phrase *return code* categorizes many of these routine messages, often a number keyed to a corresponding message. The returned code may be a phrase or sentence, but even then, the word meanings are typically application-specific and therefore still require knowledge beyond normal English. Application developers should use all available return codes, and continue to study new software releases to incorporate newly available messages into the program. Besides regular event messages, robust applications predict and prevent errors, and they also respond and react. If necessary, they finish as well as possible. Microsoft Corporation (2008b) states this error recovery goal succinctly:

> Well-written applications include error-handling code that allows them to recover gracefully from unexpected errors. When an error occurs, the application may need to request user intervention, or it may be able to recover on its own. In extreme cases, the application may log the user off or shut down the system.

Any application can receive return messages from the programming language, the operating system, and from itself, in these three categories:

- Return messages from the programming framework (such as SAS) including whether a command successfully completed, and sometimes how well it completed or failed.
- Return messages from the operating system (such as Windows) resulting from issuing an operating system command.
- Return messages from an application are explicitly programmed to result from running a function, method, or command within the operating environment.

The purpose of this paper is to describe how these three return code categories support robust applications development.

RETURN CODES FROM SAS

SAS 9.1.3 provides return codes in different feature categories. This section will summarize handling return messages using seven feature categories:

1. System Options
2. Automatic Macro Variables
3. ARM (Application Response Measurement) Macros
4. Functions
5. Call MODULE Routine
6. PUT Statement
7. SCL Event Classes

SYSTEM OPTIONS

SAS Institute (2007e) defines *system option*:

> *System options* are instructions that affect your SAS session. They control the way that SAS performs operations such as SAS System initialization, hardware and software interfacing, and the input, processing, and output of jobs and SAS files.

The SAS session includes return codes and messages that developers can use in applications. The SAS documentation provides specific instruction on how to obtain system option information and how to use them (SAS Institute Inc., 2007f).

Table 1.
SAS System Options Important for Return Code Processing

Category	SAS System Options	Description
Environment control: Error handling	BYERR System Option	Controls whether SAS generates an error message and sets the error flag when a _NULL_ data set is used in the SORT procedure
	CLEANUP System Option	Specifies how to handle an out-of-resource condition
	DMSSYNCHK System Option	Enables syntax check mode for multiple steps in the SAS windowing environment
	DSNFERR System Option	Controls how SAS responds when a SAS data set is not found
	ERRORABEND System Option	Specifies how SAS responds to errors
	ERRORBYABEND System Option	Specifies how SAS responds to BY-group error conditions
	ERRORCHECK= System Option	Controls error handling
	ERRORS= System Option	Controls the maximum number of observations for which complete error messages are printed
	FMTERR System Option	Determines whether SAS generates an error message when a format of a variable cannot be found
	QUOTELENMAX System Option	Specifies that SAS write to the SAS log a warning about the maximum length for strings in quotation marks
	SYNTAXCHECK System Option	Enables syntax check mode for multiple steps in noninteractive or batch SAS sessions
	VNFERR System Option	Controls how SAS responds when a _NULL_ data set is used
Environment control: Files	NEWS= System Option	Specifies a file that contains messages to be written to the SAS log
Files: SAS Files	DATASTMTCHK= System Option	Prevents certain errors by controlling the SAS keywords that are allowed in the DATA statement
	DKRICOND= System Option	Controls the error detection for input data sets during processing of DROP=, KEEP=, and RENAME= data set options
	DKROCOND= System Option	Controls the error detection for output data sets during processing DROP=, KEEP=, and RENAME= data set options and the corresponding DATA step statements
	DLDMGACTION= System Option	Specifies what action to take when a SAS catalog or a SAS data set in a SAS data library is detected as damaged
	MERGENOBY System Option	Controls whether a message is issued when MERGE processing occurs without an associated BY statement
Input control: Data Processing	INVALIDDATA= System Option	Specifies the value SAS is to assign to a variable when invalid numeric data are encountered
Log and procedure output control: SAS log	ECHOAUTO System Option	Controls whether autoexec code in an input file is echoed to the log
	LOGPARM= System Option	Controls when SAS log files are opened, closed, and, with the LOG= system option, how they are named
	MSGLEVEL= System Option	Controls the detail in messages that are written to the SAS log
	NOTES System Option	Writes notes to the SAS log
	OVP System Option	Overprints output lines
	PRINTMSGLIST System Option	Controls printing extended lists of messages to the SAS log
	SOURCE System Option	Controls whether SAS writes source statements to the SAS log
	SOURCE2 System Option	Controls whether SAS writes secondary source statements from included files to the SAS log
Macro: SAS macro	MACRO System Option	Specifies whether the SAS macro language is available
	MCOMPILENOTE= System Option	Issues a NOTE to the log upon successful compilation of a macro
	MERROR System Option	Controls whether SAS issues a warning message when a macro-like name does not match a macro keyword
	MFILE System Option	Specifies whether MPRINT output is directed to an external file
	MLOGIC System Option	Controls whether SAS traces execution of the macro language processor
	MLOGICNEST System Option	Displays macro nesting information in the MLOGIC output in the SAS log
	MPRINT System Option	Displays SAS statements that are generated by macro execution
	MPRINTNEST System Option	Displays macro nesting information in the MPRINT output in the SAS log
	SERROR System Option	Controls whether SAS issues a warning message when a defined macro variable reference does not match a macro variable
	SYMBOLGEN System Option	Controls whether the results of resolving macro variable references are written to the SAS log
System administration: Performance	ARMSUBSYS= System Option	Enables and disables the ARM subsystems that determine the internal SAS processing transactions to be logged
	CMPOPT= System Option	Specifies the code generation optimizations to use in the SAS language compiler

By design, system options remain active over several tasks or even an entire SAS session. Sometimes, applications may need to set or reset certain options during processing. Querying system options is available through read-only dictionary tables (SAS Institute Inc., 2005a):

> A DICTIONARY table is a read-only SAS data view that contains information about SAS data libraries, SAS data sets, SAS macros, and external files that are in use or available in the current SAS session. A DICTIONARY table also contains the settings for SAS system options that are in effect. When you access a DICTIONARY table, SAS determines the current state of the SAS session and returns the desired information accordingly. This process is performed each time a DICTIONARY table is accessed, so you always have current information.
>
> DICTIONARY tables can be accessed by a SAS program by using either of these methods:
>
> - run a PROC SQL query against the table, using the DICTIONARY libref
> - use any SAS procedure or the DATA step, referring to the PROC SQL view of the table in the SASHELP library.

The SAS documentation considers DICTIONARY tables as part of the SQL procedure (SAS Institute Inc., 2006a), though they can be read as dictionary views outside the SQL procedure. The SAS documentation provides a good overview on how system options are stored in these special SAS datasets (SAS Institute Inc., 2003e). Many recent papers provide excellent, detailed instruction on how to use this important tool for applications development (Davis, 2001; DiIoio & Abolafia, 2004; Eberhardt & Brill, 2006; Lafler, 2005).

Because dictionary tables are read-only, the feature cannot be used to set options. System options could be set with an OPTIONS statement. However, this paper recommends using the SCL commands OPTGETC, OPTGETN, OPTSETC and OPTSETN, which each have return codes for successful completion.

AUTOMATIC MACRO VARIABLES

SAS Institute (2003c) provides a good introduction to macro variables. Later, the documentation describes the role and scope of *automatic macro variables* (SAS Institute Inc., 2003d):

> When you invoke SAS, the macro processor creates *automatic macro variables* that supply information related to the SAS session. Automatic variables are global except SYSPBUFF, which is local. Automatic macro variables are often useful in conditional logic such as a %IF statement with actions determined by the value that is returned. %IF is described in Macro Language Dictionary… You can assign values to automatic macro variables that have read and write status. However, you cannot assign a value to an automatic macro variable that has read-only status... Use %PUT _AUTOMATIC_ to view all available automatic macro variables. There are also system-specific macro variables that are created only on a particular platform.

Automatic macro variables act as complement to the system options (Markovitz, 2002). While system options set the state of the SAS session, the automatic macro variables allow the SAS session to report its state. The writeable automatic macro variables allow the developer to store state information. Though all automatic macro variables help build a robust application, the following table includes those of particular interest to generic return code processing.

Table 2.
Automatic Macro Variables Important for Return Code Processing

Status	Variable	Contains
Read and Write	SYSRC	various system-related return codes
read-only	SYSPROCESSID	the process id of the current SAS process
	SYSPROCESSNAME	the process name of the current SAS process

ARM (APPLICATION RESPONSE MEASUREMENT) MACROS

SAS Institute (2007d) defines the term *ARM macros*:

> The ARM [Application Response Measurement] macros provide a way to measure the performance of applications as they are executing. The macros write transaction records to the ARM log. The ARM log is an external output text file that contains the logged ARM transaction records. You insert the ARM macros into your SAS program at strategic points in order to generate calls to the ARM API [Application Programming Interface] function calls. The ARM API function calls typically log the time of the call and other related data to the log file. Measuring the time between these function calls yields an approximate response-time measurement. An ARM macro is self-contained and does not affect any code surrounding it, provided that the variable name passed as an option to the ARM macro is unique. The ARM macros are used in open code (code that is not in PROC or DATA steps) and in DATA step or SCL environments… The ARM macros are not part of SAS Macro Facility. They are part of the SAS ARM interface.

The SAS documentation provides instruction on monitoring performance using ARM macros, including what it is and how it works (SAS Institute Inc., 2005b). SAS describes ARM macros in version 9 as a manageability feature to check the availability and transaction rates of SAS applications (SAS Institute Inc., 2006b, p. 7). The topic deserves its own paper and tutorial (Coffey, 2003). The ARM macros are activated with a SAS system option, and the information is accessed through macros run between data steps or procedures. These functions could be coded through other means, specifically by using the SAS macro facility. However, the ARM macros have been optimized to run quickly alongside an application, therefore providing minimum performance burden on the process. The information is stored in global macro variables, but ARM functions allow for transferring the data from ARM macro variables to application variables or macro variables. Transferring the information becomes important when using nested macros, SCL methods, and SAS/AF objects. The current ARM framework makes them best suited for monitoring data step and SAS procedure submissions. Application developers should be aware of any enhancements SAS makes to the ARM macros.

FUNCTIONS

SAS Institute (2007b) defines the term *function*:

> A *SAS function* performs a computation or system manipulation on arguments and returns a value. Most functions use arguments supplied by the user, but a few obtain their arguments from the operating environment. In Base SAS software, you can use SAS functions in DATA step programming statements, in a WHERE expression, in macro language statements, in PROC REPORT, and in Structured Query Language (SQL). Some statistical procedures also use SAS functions. In addition, some other SAS software products offer functions that you can use in the DATA step. Refer to the documentation that pertains to the specific SAS software product for additional information about these functions.

The SAS documentation provides instruction on how to use functions (SAS Institute Inc., 2007c). Some SAS functions moderate how an application can use return messages.

Table 3.
SAS Functions Important for Return Code Processing

Category	Functions	Description
External Files	SYSMSG Function	Returns the text of error messages or warning messages from the last data set or external file function execution
	SYSRC Function	Returns a system error number
Macro	SYMEXIST Function	Returns an indication of the existence of a macro variable
Special	GETOPTION Function	Returns the value of a SAS system or graphics option
	SYSGET Function	Returns the value of the specified operating environment variable
	SYSPROCESSID Function	Returns the process id of the current process
	SYSPROCESSNAME Function	Returns the process name associated with a given process id or the name of the current process
	SYSPROD Function	Determines if a product is licensed
	SYSTEM Function	Issues an operating environment command during a SAS session and returns the system return code

Application developers can use these functions within data steps or in SAS/AF code to provide applications with information on system processing. Besides the listed functions, many SCL functions also have return codes useful in robust applications development.

CALL MODULE ROUTINE

SAS Institute (2007b) defines the term *CALL routines*:

> A *CALL routine* alters variable values or performs other system functions. CALL routines are similar to functions, but differ from functions in that you cannot use them in assignment statements. All SAS CALL routines are invoked with CALL statements; that is, the name of the routine must appear after the keyword CALL on the CALL statement.

The SAS documentation provides instruction on the syntax for CALL routines, one of which is CALL module, which "calls the external routine without any return code" (SAS Institute Inc., 2007a). Because a CALL module routine may have any number of input, updated, or return fields among its arguments, a CALL module actually "calls the external routine without an *explicit or expected* SAS return code". Logically, SAS cannot automatically return a code since it is unaware of what to expect from every CALL module routine. The SAS session might know the types of variables passed (such as character or numeric) but will not be able to categorize a CALL as being successful or not.

The CALL module routine allows a user to call the Windows API, and generically any DLL (dynamic linked library, a COM executable) programmed in Windows (SAS Institute Inc., 2003f). In like manner, SAS running on Unix allows the CALL module access to shared executable libraries (SAS Institute Inc., 2003j). In each case, these modules can provide return messages which may help build a more robust application.

PUT STATEMENT

The PUT statement is one of the most versatile tools for reporting on return codes. Used in a SAS session, the statement can allow for reporting values from inside a data step. Also, the statement can output either SAS-defined or user-defined macro variables. The output destination can be the log or any other ODS (output delivery system) location. The PUT statement can apply a SAS format to the output. SAS Institute (2007k) provides extended information on this versatile reporting tool. Also, the SAS documentation instructs on how to use statements (SAS Institute Inc., 2007j). PUT statements are not the only way to report errors, and instead (or in addition) a developer might choose to write information to a SAS dataset as a form of event logging.

SCL EVENT CLASSES

Details of object-oriented programming are beyond the scope of this paper. SCL event classes can be used to effectively communicate among components (SAS Institute Inc., 2003i). Events allow communication from and among visual and nonvisual objects in SAS/AF applications. Developers should be cautioned that SAS implements global event registration, which might have implications for a SAS session running multiple applications in the same process (SAS Institute Inc., 2004). These implications include assuring a unique event naming system, and also being careful for event handlers programmed to catch any senders.

Previously experienced developers may choose to implement the SCL Exception class, which inherits from the SCL Throwable class (SAS Institute Inc., 2002; Scocca, 2004). As with .NET languages, SAS exception handling is a specific implementation of event handling, where exceptions (events) are thrown to exception (event) handlers. *Exceptions* are properly considered to be errors or potential errors in application code. Thus, to reduce maintenance ambiguity, developers should not be using exception coding for regular events.

SUMMARY

The following table summarizes what SAS provides natively across platforms for return code messages, and for catching errors:

Table 4.
Summary of SAS Features Important for Return Code Processing

Category	Description	URL Reference (Retrieved February 1, 2008)
System Options	Sets choices for the SAS session state	http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000309878.htm (SAS Institute Inc., 2007i)
Automatic Macro Variables	Provides information on the system state	http://support.sas.com/onlinedoc/913/getDoc/en/mcrolref.hlp/a002047074.htm (SAS Institute Inc., 2003k)
ARM Macros	Provides a lightweight structure for measuring and monitoring application performance	http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a002295646.htm (SAS Institute Inc., 2007h)
Functions	Provides interactive information on current processing	http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000245852.htm (SAS Institute Inc., 2007g)
CALL Module Routine	Allows for accessing modules though Windows and Unix	http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000471253.htm (SAS Institute Inc., 2007a)
PUT Statement	General reporting tool	http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000161869.htm (SAS Institute Inc., 2007k)
SCL Event Classes	Inheritable class for error reporting	http://support.sas.com/onlinedoc/913/getDoc/en/sclref.hlp/a001090274.htm (SAS Institute Inc., 2003i)

Robust applications probably will use features in these categories beyond those listed, but the focus of this paper are the essential return code features for a SAS session. The next section covers Windows specifically, and shows how SAS works with Windows to provide return messages important for handling errors.

RETURN CODES FROM WINDOWS

Though SAS runs on many platforms, this paper focuses on Windows as a typically used platform. Assuming SAS 9.1.3 running on the Windows platform, the applications developer has the choice of return messages in these categories:

1. SAS System Options for Windows
2. SAS Automatic Macro Variables for Windows
3. SAS Functions and CALL Routines for Windows
4. Windows 32 API System Error Codes

The first three categories are extensions to those mentioned in the first section of this paper. The SAS documentation includes a separate platform-specific area of documentation which includes commands not in the general lists. The next table provides the URL (uniform resource locator) for these additional system options, automatic macro variables, functions, and CALL routines for Windows:

Table 5.
SAS Documentation on Windows-Specific Features Important for Return Code Processing

Category	Description	URL Reference (Retrieved February 1, 2008)
System Options for Windows	Sets choices for the SAS session state	http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/sysopts.htm (SAS Institute Inc., 2003h)
Automatic Macro Variables for Windows	Provides information on the system state	http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/automacrovar.htm (SAS Institute Inc., 2003b)
Functions and CALL Routines for Windows	Provides interactive information on current processing	http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/function.htm (SAS Institute Inc., 2003g)

The fourth category is *Windows 32 API System Error Codes,* which a SAS session can obtain through a CALL module command. These error codes from *kernel32.dll* work on 32-bit Windows, and an identical alternative version continues to be available in 64-bit editions for Vista and Windows Server 2008 (Pietrek, 2006). Using a CALL routine, a SAS application can set, update, or retrieve information from the Windows session (SAS Institute Inc., 2003f). Beyond just error codes, several SAS users have published papers on this general Windows API topic, notably including my Australian colleague David Johnson (Johnson, 2001, 2003, 2005; Woo, 2006).

When SAS runs in Windows (or any operating system) there are only a subset of commands available for return code processing, and as much as possible, work happens within the SAS COM session. For example, many database and disk functions available through the Windows 32 API are best called through SAS functions because SAS can fully and simply provide an appropriate return code. In cases when SAS does not provide full or sufficient Windows capability, calling a Windows DLL may provide a better solution. After calling a COM executable (typically a DLL), it is important to interpret what Windows calls *system error codes*. Unlike the comparatively parsimonious tables of SAS error codes, the Microsoft documentation lists thousands of available codes (Microsoft Corporation, 2008f). SAS can retrieve these codes through the *GetLastError* function (Microsoft Corporation, 2008c).

SAS 9.1.3 on Windows continues to run under what Microsoft calls *automation* (Microsoft Corporation, 2008a). SAS provides a COM interface using a COM subset known as *OLE* (SAS Institute Inc., 2003a). Microsoft applications in Visual Basic or a .NET language (such as Visual Basic .NET or C# .NET) can access a SAS session using automation. In reverse, SAS sessions can access COM interfaces (such as *kernel32.dll*) directly (SAS Institute Inc., 2003f). Programs written in .NET can provide COM access through the Assembly Registration Tool (Microsoft Corporation, 2008e). Through this tool, SAS programs can have access not only to regular events but also to the rich exception classes available in .NET 1.0, 1.1, 2.0, and 3.5, both from native Windows and also from customized .NET development (Microsoft Corporation, 2008b; SAS Institute Inc., 2008a, 2008b, 2008c, 2008d). For example, using .NET and assembly registration, SAS developers could participate in Windows Error Reporting (Microsoft Corporation, 2008d). More normally, developers may choose to tie .NET event messaging to their SAS applications.

When considering how to build a robust application, consider the SAS document specific to that operating environment. Additional features, such as functions, automatic macro variables, and system options may allow for additional tools for error checking and handling. Using COM (and specifically OLE), the Windows operating system provides many error checking tools and features available to SAS applications. Though only Windows is presented, please see the SAS operating system documentation for options on other platforms (including Unix and MVS).

YOUR RETURN CODES

This paper assumes that an *application* includes subroutines, implemented as methods or script modules. The methods might be written in SAS/AF or a .NET language or Java. The script module might be a SAS macro or Javascript. Complex applications might use object-oriented and scripting languages to achieve the task. The whole scope of this complexity is bundled into what this paper calls an *application*, whether it is one single code structure or a federation of communicating code structures. Event programming extends communication possibilities across components or whole applications.

In any application conversation, there are three mutually distinct outcomes after issuing a command:

1. Receive no response
2. Receive a response sometimes
3. Always receive a response

Skipping to the answer, the third case is best, when the language or operating system or application always returns an adequately thorough explanation. In case number one, it is not possible to tell whether a command issued was successful. No response is an indication of a poor language (command), or operating system, or application. The

second case might involve a complex response, where the code is more than just a binary successful or not, but might include nuanced flavors of success. In a typical programming environment, all three cases are present. By proactively creating your own customized return codes, messages, and framework, you can supplement what is available and improve your application. Providing specific messages enhances the user experience, improves application reporting, and in the unlikely event of an error, provides developers with data for reducing future problems.

The previous sections showed what SAS can provide, and what SAS and Windows can provide together. Based on concepts and categories in previous sections, table six provides ideas about how to develop your own return codes.

Table 6.
Customized Extensions for Return Code Processing

Category	Description
Application Options	Similar to SAS system options, application options are variables declared either globally or locally for an application, and allow for return code processing to be controlled from a higher level. The variables could be stored in application datasets, similar to the read-only dictionary tables.
Application Macro Variables	Similar to SAS automatic macro variables, the application macro variables (global or local) provide the ability to keep the state at a specific point. Information stored could be character or text (though macro variables always store information as text).
Application Macros	Similar to SAS ARM Macros, the application can have its own macros to monitor and measure performance. These customized macros could extend the capabilities of ARM macros, providing the application with robust reporting on performance. Developing these macros separately from other macros allows them to be used in future applications.
Functions	Similar to SAS functions, applications can define methods in SCL. These methods can provide a return code, and could extend or incorporate what SAS natively provides for return code processing. Encapsulating these methods individually, or as a group (creating a class or classes using SAS/AF) would allow these methods or classes to be copied into future applications.
CALL Module Routine	Similar to calling Windows DLLs, SAS applications running in Windows could call programmed .NET DLLs. These modules could have return codes and messages helping the application to become more robust. Generically developed DLLs have the advantage of being used across many custom applications.
Reporting Datasets and Forms	Instead of simply using the PUT statement to send variable values to the log or ODS destination, an application could send the information into a dataset, which could include the computer name and time and date of logging. Also, the application could have customized forms for reporting errors, including the option to send those reports as an attachment or the body of an e-mail.
Event Classes	Create event classes which capture specific return codes as events. Create customized SCL classes based on the SCL exception class; alternatively, use a language which can produce a COM interface to work with SAS to provide advanced error capabilities. Use event classes to communicate across components and applications.

Table six provides categorized ideas for actively creating and developing a robust return code processing system alongside a current application. Robust applications participate also in the active and dynamic process of preparing and delivering customized return code messages. These extensions might be parsimoniously implemented as binary return codes, but they also could be as complex as fully formatted e-mails that include explanatory text and graphics.

Based on the SAS and Windows development structure, the following bullet points provide advice for reusable customized return messaging development:

- Use binary codes for success or failure
- When more than binary codes are returned, consider using globally enumerated macro variables, global SCL variables, global SCL lists, or SAS read-only datasets to allow for named references to return code conditions
- Use simple, reusable names for macros, functions, and classes
- Code for the complete set of possible conditions, not only the ones proven through experience
- Optionally implement reporting to the SAS log, or to an external event log (database or file)

When developing a reusable library of return code features and mechanisms, a generically developed return messaging framework could be used across applications. Common features of SAS and Windows event messaging and error processing provide insight into best practices for customized return code development. By keeping the commands simple and the messages general, these same tools can be used across many applications. Specific or complex reports might make sense for specific applications, but will not have generalized use across a library of future customized code. In object-oriented applications, it is best to allow base event and reporting features to remain in parent classes, and have specific return code features implemented through overloading and overwriting methods. Another way is to use arrays, SCL lists, or datasets to provide customization to classes or base SAS code. The main point is to brainstorm creatively on what optimal solution makes sense for a given application. The SAS system provides the technology to fully implement a variety of application designs.

CONCLUSION

This paper outlined three major categories of return code processing in SAS application. The first category included

what SAS provides directly. The second category included what SAS provides with Microsoft Windows. The last category covered what the applications developer can provide when linking multiple languages and applications. Robust applications not only predict and prevent errors, but they also respond and react, and when necessary, terminate as gracefully as possible. Developers who incorporate newly available return codes as well as code better return messages improve their own understanding of not only what is available in programming languages, but also provides insight into how to improve both languages and applications.

REFERENCES

Coffey, M. (2003). Using the SAS® v 9 Application Response Measurement system to provide metrics to HP-UX Workload Manager. *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi28/290-28.pdf

Davis, M. (2001). You could look it up: An introduction to SASHELP dictionary view. *Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi26/p017-26.pdf

DiIoio, F., & Abolafia, J. (2004). Dictionary Tables and Views: Essential Tools for Serious Applications. *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi29/237-29.pdf

Eberhardt, P., & Brill, I. (2006). How do I look it up If I cannot spell it: An introduction to SAS® dictionary tables. *Proceedings of the Thirty-First Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi31/259-31.pdf

Johnson, D. (2001). Coding across the boundaries. *Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi26/p280-26.pdf

Johnson, D. (2003). Multi-platform SAS®, Multi-platform code. *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi28/001-28.pdf

Johnson, D. (2005). SAS® with the Windows API. *Proceedings of the Thirtieth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi30/248-30.pdf

Lafler, K. P. (2005). Exploring DICTIONARY tables and views. *Proceedings of the Thirtieth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi30/070-30.pdf

Markovitz, H. (2002). SAS Completion Codes to Make Complex Programs Run Smoothly. *Proceedings of the Southeast SAS Users' Conference* Retrieved February 1, 2008, from http://analytics.ncsu.edu/sesug/2002/CC07.pdf

Microsoft Corporation. (2008a). Automation. *MSDN® Library* Retrieved February 1, 2008, from http://msdn2.microsoft.com/en-us/library/dt80be78.aspx

Microsoft Corporation. (2008b). Error handling (Windows). *MSDN® Library* Retrieved February 1, 2008, from http://msdn2.microsoft.com/en-us/library/ms679320.aspx

Microsoft Corporation. (2008c). GetLastError Function. *MSDN® Library* Retrieved February 1, 2008, from http://msdn2.microsoft.com/en-us/library/ms679360(VS.85).aspx

Microsoft Corporation. (2008d). Introducing Windows Error Reporting. Retrieved February 1, 2008, from http://msdn2.microsoft.com/en-us/isv/bb190483.aspx

Microsoft Corporation. (2008e). .NET Framework Tools: Assembly Registration Tool (Regasm.exe). *MSDN® Library* Retrieved February 1, 2008, from http://msdn2.microsoft.com/en-us/library/tzat5yw6.aspx

Microsoft Corporation. (2008f). System error codes. *MSDN® Library* Retrieved February 1, 2008, from http://msdn2.microsoft.com/en-us/library/ms681381.aspx

Pietrek, M. (2006). x64 primer: Everything you need to know to start programming 64-bit Windows systems. *MSDN® Magazine* Retrieved February 1, 2008, from http://msdn.microsoft.com/msdnmag/issues/06/05/x64/default.aspx

SAS Institute Inc. (2002). Using the SCL Exception Class. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/af.hlp/usesclexception.htm

SAS Institute Inc. (2003a). Controlling SAS from Another Application Using OLE under Windows: Introduction to Automating SAS. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/introoleauto.htm

SAS Institute Inc. (2003b). Macro Facility under Windows: Automatic Macro Variables. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/automacrovar.htm

SAS Institute Inc. (2003c). Macro Variables: Introduction to Macro Variables. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/mcrolref.hlp/a002293823.htm

SAS Institute Inc. (2003d). Macro Variables: Macro Variables Defined by SAS. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/mcrolref.hlp/a001071968.htm

SAS Institute Inc. (2003e). Programming with the SQL Procedure: Accessing SAS System Information Using DICTIONARY Tables. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/sqlproc.hlp/a001385596.htm

SAS Institute Inc. (2003f). SAS 9.1 Companion for Windows: Accessing External DLLs from SAS under Windows. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/accdll.htm

SAS Institute Inc. (2003g). SAS 9.1 Companion for Windows: Functions and CALL Routines under Windows. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/function.htm

SAS Institute Inc. (2003h). SAS 9.1 Companion for Windows: System Options under Windows. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/hostwin.hlp/sysopts.htm

SAS Institute Inc. (2003i). SAS Object-Oriented Programming Concepts: Events and Event Handlers. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/sclref.hlp/a001090274.htm

SAS Institute Inc. (2003j). SAS® 9.1 Companion for UNIX Environments: Accessing Shared Executable Libraries from SAS. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/hostunx.hlp/unxsharedlib.htm

SAS Institute Inc. (2003k). SAS® 9.1 Macro Language: Reference: Macro Variables. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/mcrolref.hlp/a002047074.htm

SAS Institute Inc. (2004). Adding Communication Capabilities to Components: Modifying or Adding Event Handling. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/appdevgd.hlp/a000934720.htm

SAS Institute Inc. (2005a). DICTIONARY tables: Definition of a DICTIONARY table. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrcon.hlp/a002300184.htm

SAS Institute Inc. (2005b). SAS® 9.1.3 Language Reference: Concepts, Third Edition: Monitoring Performance Using Application Response Measurement (ARM). *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrcon.hlp/a002212999.htm

SAS Institute Inc. (2006a). Base SAS® 9.1.3 Procedures Guide: The SQL Procedure. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/proc.hlp/a000086336.htm

SAS Institute Inc. (2006b). What's New: SAS 9.0, 9.1, 9.1.2, and 9.1.3 Highlights. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/whatsnew.hlp/whatsnewoverview.htm

SAS Institute Inc. (2007a). Functions and CALL Routines: CALL MODULE Routine. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000471253.htm

SAS Institute Inc. (2007b). Functions and CALL Routines: Definitions of Functions and CALL Routines. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a001131396.htm

SAS Institute Inc. (2007c). Functions and CALL Routines: Using Functions. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a001281551.htm

SAS Institute Inc. (2007d). SAS ARM Macros: Definition of ARM Macros. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a002198722.htm

SAS Institute Inc. (2007e). SAS System Options: Definition of System Options. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000103940.htm

SAS Institute Inc. (2007f). SAS System Options: Using SAS System Options. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000103941.htm

SAS Institute Inc. (2007g). SAS® 9.1.3 Language Reference: Dictionary, Fifth Edition: Functions and CALL Routines. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000245852.htm

SAS Institute Inc. (2007h). SAS® 9.1.3 Language Reference: Dictionary, Fifth Edition: SAS ARM Macros. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a002295646.htm

SAS Institute Inc. (2007i). SAS® 9.1.3 Language Reference: Dictionary, Fifth Edition: SAS System Options. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000309878.htm

SAS Institute Inc. (2007j). SAS® 9.1.3 Language Reference: Dictionary, Fifth Edition: Statements. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000293668.htm

SAS Institute Inc. (2007k). Statements: PUT Statement. *SAS OnlineDoc 9.1.3 for the Web* Retrieved February 1, 2008, from http://support.sas.com/onlinedoc/913/getDoc/en/lrdict.hlp/a000161869.htm

SAS Institute Inc. (2008a). Programming in the .NET Environment. *Knowledge Base/Product Documentation* Retrieved February 1, 2008, from http://support.sas.com/rnd/itech/doc9/dev_guide/dist-obj/winclnt/windotnet.html

SAS Institute Inc. (2008b). Sample 25272: ASP.NET Sample Library. *Knowledge Base/Samples & SAS Notes* Retrieved February 1, 2008, from http://support.sas.com/kb/25/272.html

SAS Institute Inc. (2008c). Sample 25276: Reading SAS Data in .Net Using ADO and ADO.NET. *Knowledge Base/Samples & SAS Notes* Retrieved February 1, 2008, from http://support.sas.com/kb/25/276.html

SAS Institute Inc. (2008d). Sample 25959: ASP.NET Web Service. *Knowledge Base/Samples & SAS Notes* Retrieved February 1, 2008, from http://support.sas.com/kb/25/959.html

Scocca, D. A. (2004). Building A More Robust SAS® Application. *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi29/038-29.pdf

Woo, W. (2006). Using SAS/CONNECT® and the Windows API to Preserve File DateTime Stamps in a Platform Migration Project. *Proceedings of the Thirty-First Annual SAS Users Group International Conference* Retrieved February 1, 2008, from http://www2.sas.com/proceedings/sugi31/179-31.pdf

TRADEMARK CITATION

CONTACT INFORMATION

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

Mark Tabladillo
Web: http://www.marktab.com