



Dear Readers,

SAS is always looking for ways to connect with our users. For that reason, we are excited to announce [user discussion forums](#). Visit our new forum site to participate in focused discussions that are sure to include a variety of tips, questions, and problem solutions. Choose from ODS and base reporting, integration with Microsoft Office, SAS stored processes and more.

In addition to talking with users, we are honoring longtime SAS users with a new program called the SAS Silver Circle. We invite all SAS users who have 25 years or more experience with our software to share their stories. All submissions meeting the minimum longevity requirement will be included in the SAS Silver Circle. A select group also will be highlighted in future stories on the SAS Web site, **sascom** magazine and at SAS events. Find out more on the [Web](#). And be sure to throw your name in the ring!

Thanks for your dedication to SAS.

A handwritten signature in black ink that reads "Shelley Sessoms".

[Shelley Sessoms](#)

Editor, *Your SAS Technology Report*

An Introduction to Exporting SAS/Graph Output to Microsoft Office

SAS Release 8.2 and higher

This document includes in-depth information about the different formats and methods you can use to export SAS/GRAPH output to Microsoft Office 97, 2000, and XP. This document has been updated to include new features that are now available in SAS 9. Throughout the document, logos are used to indicate where new features are available in SAS 9.0 or SAS 9.1.

View the entire multi-page document at

<http://support.sas.com/techsup/technote/ts674/ts674.html>

FAQ # 1797

Q: I need to hardcode my user ID and password within my script file on the client so that I can sign on in batch mode. What is the best way to protect the script file after I have added my user ID and password?

A: To pass the user ID and password without hardcoding them within the script file, see [FAQ # 1800](#). Otherwise, to protect the script file when hardcoding the user ID and password within the file, use the access controls that are provided by your client operating system.

- On UNIX, you can set the file mode access permissions.
- On MVS, you can use RACF.
- On Windows NT, Windows 2000, and Windows XP, you have access controls when using NTFS only.
- Windows 9x and Windows Me do not provide access controls.

For more details, refer to your operating system documentation.

SAS 9.1.3 Service Pack 4 is Available

SAS 9.1.3 Service Pack 4 is available on a limited basis for SAS 9.1.3 customers. You can download the Service Pack for all operating systems except OpenVMS Alpha. You can request Service Pack 4 on media for z/OS (OS/390) only; media will be available for other operating systems at a later date. Service Pack 4 includes the fixes and enhancements from Service Packs 1, 2 and 3, as well as additional fixes and enhancements.

[Request the Service Pack media](#)

[Download the Service Pack](#)

Using Regular Expressions

Perl Regular Expressions are supported beginning with Version 9.

Introduction

This page will discuss uses of Perl Regular Expressions in the DATA step. Three typical uses of regular expressions will be discussed and an example will be presented for each.

Regular expressions, or regexp, are used to search text. Regexp may seem foreign if you have not used them before. But, you may have encountered them without realizing it. Think about how you might list items in a directory. Often times the ? and * characters are used to match one character or zero or more characters. Performing the command `ls data*.txt` or `dir data*.txt` from a shell prompt would list the files that begin with data and end in .txt. The * character doesn't search for a star, but for zero or more characters. Characters like * are where the power of regexp come from. Perl Regexp build on ? and * to make searching within text powerful and flexible.

Uses

Regexp enhance search and replace operations on text. From the DATA step, the INDEX and SUBSTR functions along with concatenation (||) can be used for simple search and replace operations on static text. These functions lack flexibility and make searching dynamic text difficult, if not impossible.

Regexp can be used to:

1. Test for a pattern of characters within a string. For example, a string can be examined to determine if it contains a correctly formatted telephone number. This is called data validation.
2. Replace text. Regexp can be used to find specific text within a string. The found text can be removed or replaced with other text.
3. Extract a substring from a string. Regexp can be used to find and easily extract text found within a string.

For each of the examples below, equivalent code could be written that does not use regexp. The non-regexp code would involve more function calls, dealing with character positions in a string, and manipulating parts of the string. Regexp combines most, if not all, of these steps into one expression. This makes code less error prone, easier to maintain, more clear, and can improve performance.

Syntax

Regexp are composed of characters and special characters called metacharacters. When performing a regexp match, a character matches that character in the string. The metacharacters perform special actions when matching, like forcing the match to begin at a particular location or by matching a particular set of characters. The following table contains common regexp metacharacters and will come in handy when building and understanding

regexp. A complete list can be found in the [perlre man page](http://search.cpan.org/~gsar/perl-5.6.1/pod/perlre.pod) (<http://search.cpan.org/~gsar/perl-5.6.1/pod/perlre.pod>).

Meta character	Description
\	Marks the next character as either a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\' matches "\" and \"(\" matches "(".
^	Matches the position at the beginning of the input string.
\$	Matches the position at the end of the input string.
*	Matches the preceding subexpression zero or more times. For example, zo* matches "z" and "zoo". * is equivalent to {0,}.
+	Matches the preceding subexpression one or more times. For example, 'zo+' matches "zo" and "zoo", but not "z". + is equivalent to {1,}.
?	Matches the preceding subexpression zero or one time. For example, "do(es)?" matches the "do" in "do" or "does". ? is equivalent to {0,1}
{n}	n is a nonnegative integer. Matches exactly n times. For example, 'o{2}' does not match the 'o' in "Bob," but matches the two o's in "food".
{n,}	n is a nonnegative integer. Matches at least n times. For example, 'o{2,}' does not match the "o" in "Bob" and matches all the o's in "fooooo". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'.
{n,m}	m and n are nonnegative integers, where n <= m. Matches at least n and at most m times. For example, "o{1,3}" matches the first three o's in "fooooo". 'o{0,1}' is equivalent to 'o?'. Note that you cannot put a space between the comma and the numbers.
.	Matches any single character except "\n". To match any character including the '\n', use a pattern such as '[.\n]'.
(pattern)	Matches pattern and captures the match. The position and length of the captured match can be retrieved by using CALL PRXPOSN (PRX POSition). To match parentheses characters (), use \"(' or \)'.
x y	Matches either x or y. For example, 'z food' matches "z" or "food". '(z f)ood' matches "zood" or "food".
[xyz]	A character set. Matches any one of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
[^xyz]	A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
[a-z]	A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
[^a-z]	A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
\b	Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".
\B	Matches a nonword boundary. 'er\B' matches the 'er' in "verb" but not the 'er' in "never".

<code>\d</code>	Matches a digit character. Equivalent to <code>[0-9]</code> .
<code>\D</code>	Matches a nondigit character. Equivalent to <code>[^0-9]</code> .
<code>\s</code>	Matches any whitespace character including space, tab, form-feed, etc. Equivalent to <code>[\f\n\r\t\v]</code> .
<code>\S</code>	Matches any non-whitespace character. Equivalent to <code>[^\f\n\r\t\v]</code> .
<code>\t</code>	Matches a tab character. Equivalent to <code>\x09</code> .
<code>\w</code>	Matches any word character including underscore. Equivalent to <code>[A-Za-z0-9_]</code> .
<code>\W</code>	Matches any nonword character. Equivalent to <code>[^A-Za-z0-9_]</code> .
<code>\num</code>	Matches num, where num is a positive integer. A reference back to captured matches. For example, <code>'(.)\1'</code> matches two consecutive identical characters.

Examples

1. Data Validation

Determine if there is an invalid phone number in a set of phone numbers. A phone number must be formatted as `(XXX)XXX-XXXX` or `XXX-XXX-XXXX` to be valid. There can be an optional space after the closing parenthesis on an area code and the first digit of an area code or phone number must not be a 1.

This example will be listed with its output, then picked apart in a more detailed description.

Complete Example:

```
data _null_;
  retain re;
  length first last home business $ 16;

  if _N_ = 1 then do;
    /* (XXX) XXX-XXXX */
    paren = "\([2-9]\d\d\) ?[2-9]\d\d\d\d";

    /* XXX-XXX-XXXX */
    dash = "[2-9]\d\d-[2-9]\d\d\d\d";

    /* Combine two phone patterns into one with a | */
    regexp = "(" || paren || ")( " || dash || ")";

    re = prxparse(regexp);
    if missing(re) then do;
      putlog "ERROR: Invalid regexp " regexp;
      stop;
    end;
  end;
end;
```

```

input first last home business;

if ^prxmatch(re, home) then
  putlog "NOTE: Invalid home phone number for " first last home;
if ^prxmatch(re, business) then
  putlog "NOTE: Invalid business phone number for " first last business;
datalines;
Jerome Johnson (919)319-1677 (919)846-2198
Romeo Montague 800-899-2164 360-973-6201
Imani Rashid (508)852-2146 (508)366-9821
Palinor Kent . 919-782-3199
Ruby Archuleta . .
Takei Ito 7042982145 .
Tom Joad 209/963/2764 2099-66-8474
;

```

Output:

```

NOTE: Invalid home phone number for Palinor Kent
NOTE: Invalid home phone number for Ruby Archuleta
NOTE: Invalid business phone number for Ruby Archuleta
NOTE: Invalid home phone number for Takei Ito 7042982145
NOTE: Invalid business phone number for Takei Ito
NOTE: Invalid home phone number for Tom Joad 209/963/2764
NOTE: Invalid business phone number for Tom Joad 2099-66-8474

```

Performing the same operation with INDEX and SUBSTR would be difficult and error prone because character positions would need to be tracked and manipulated. Within a single regular expression, multiple search operations are performed with one expression.

Step-by-Step:

```

data _null_;
  retain re;
  length first last home business $ 16;

  if _N_ = 1 then do;
    /* (XXX) XXX-XXXX */
    paren = "\([2-9]\d\d\) ?[2-9]\d\d-\d\d\d\d";

    /* XXX-XXX-XXXX */
    dash = "[2-9]\d\d-[2-9]\d\d-\d\d\d\d";
  end;

```

The regexp to match (XXX)XXX-XXXX is \([2-9]\d\d\) ?[2-9]\d\d-\d\d\d\d and is assigned to paren. Regexp are best understood when broken into parts:

\(Match an open paren. The \ before the (is necessary since open paren is a

metacharacter in regular expressions. In order for open paren to not take on its special meaning, a \ is placed before it. \ is often referred to as an escape character.

- [2-9] Match the digits 2 through 9. Square braces around text cause any one of the characters within the square braces to be matched. In this case, we specify a range of characters. An equivalent syntax would be [23456789].
- \d Match a digit character. \d is a metacharacter that means to match a digit. This is shorthand for [0-9].
- \d Match another digit character.
- \) Match a close paren.
- ? Match zero or one space. Notice there is a space before the question mark. Spaces are significant in regular expressions and mean to match a space in the search text. The question mark is a metacharacter that means to match zero or one of the prior character. Since the prior character is a space, this means to match zero or one spaces.

[2-9]\d\d- Knowing what you do now, this means to match a digit from 2 to 9, then two digits, then a dash, then four digits.

Breaking down the regexp for XXX-XXX-XXXX could be done in a similar way.

```
/* Combine two phone patterns into one with a | */  
regexp = "/"( " || paren || ")|( " || dash || ")"/;
```

The concatenation above combines the regexp for (XXX)XXX-XXXX and XXX-XXX-XXXX into one regexp. This enables us to search for both phone number formats from one regexp.

To do this, each regexp is placed within parens and the bar metacharacter (|) says to match either pattern. The bar metacharacter is often called the alternation character. The slashes around the entire pattern tell the regular expression compiler where the start, and more importantly, the end of the regexp are.

It is important to know where the end of the regexp is so that the user does not have to TRIM their regexp when it is compiled. Typical DATA step variables are fixed width and blank padded. Blank padding would cause a blanks to be matched in the search text. This probably is not what the user intends. This is best illustrated with the code:

```
length pattern $ 200;  
pattern = "\d\d\d";
```

Without the start and end delimiters, this regexp would match three digits, then 197 spaces. This means the text "123" would not match because it does not have 197 trailing spaces. The delimiters are required and do not have to be slashes. They must be non-alphanumeric and non-whitespace characters.

```
re = prxparse(regex);
```

When working with a regexp, it must be compiled. This is done by passing aregexp to PRXPARSE. PRXPARSE will compile the regexp and return a regexp id for the compiled pattern. Using the regexp id with other PRX functions performs operations with the compiled regexp. The regexp id is stored in a retained variable so that we don't have to continually recompile the regexp for each iteration of the DATA step.

```
if missing(re) then do;
  putlog "ERROR: Invalid regexp " regexp;
  stop;
end;
end;
```

```
input first last home business;
```

```
if ^prxmatch(re, home) then
  putlog "NOTE: Invalid home phone number for " first last home;
if ^prxmatch(re, business) then
  putlog "NOTE: Invalid business phone number for " first last business;
```

PRXMATCH takes a regexp id for a compiled regexp along with the search text and returns the position where the regexp was found in the search text. If the regexp is not found, zero is returned. In the case above, if the regexp does not find a match for the home or business phone number, a NOTE is output to the log.

2. Search and Replace

When writing HTML some characters need to be converted to their named counterpart. This example will replace occurrences of < and > in text with < and >.

The regexp for replacement is s/regular-expression/replacement-text/. The regexp is searched for and the text that it matches is replaced with the replacement text. The CALL routine PRXCHANGE is used to perform the replacement.

```
data _null_;
  retain lt_re gt_re;
  if _N_ = 1 then do;
    lt_re = prxparse('s/</&lt;');
    gt_re = prxparse('s/>/&gt;');
    if missing(lt_re) or missing(gt_re) then do;
      putlog "ERROR: Invalid regexp.";
      stop;
    end;
  end;
end;

input;
call prxchange(lt_re, -1, _infile_);
```

```

call prxchange(gt_re, -1, _infile_);

put _infile_;
datalines4;

```

The bracketing construct (...) creates capture buffers. To refer to the digit'th buffer use \<digit> within the match. Outside the match use "\$" instead of "\". (The \<digit> notation works in certain circumstances outside the match. See the warning below about \1 vs \$1 for details.) Referring back to another part of the match is called a backreference.

```

;;;

```

Output:

The bracketing construct (...) creates capture buffers. To refer to the digit'th buffer use \<digit> within the match. Outside the match use "\$" instead of "\". (The \<digit> notation works in certain circumstances outside the match. See the warning below about \1 vs \$1 for details.) Referring back to another part of the match is called a backreference.

Doing the same thing with INDEX, SUBSTR, and || to patch in the correct substitution would work, but would be more involved than using these two replacement regular expressions.

3. Extracting a Substring

Let's extend the first example and create a subset of business phone numbers which are in North Carolina. This is done by extracting the area code and checking it against a list of area codes for North Carolina.

The area code will be extracted by placing the part of the regexp that matches the area code in parenthesis. This captures that submatch. The position and length of a captured submatch can be retrieved by using CALL PRXPOSN (PRX POSitioN). The second argument to PRXPOSN is the submatch the retrieve. Submatches are counted by counting the number of open parens. The submatch for the area code will be either submatch 2 or 4.

Why 2 and 4? Let's take another look at the regexp:

```

(\([[2-9]\d\d\) ?[2-9]\d\d-\d\d\d\d)
|
([[2-9]\d\d-[2-9]\d\d-\d\d\d\d)

```

Open parens 2 and 4 are underlined and bold. This is where submatches 2 and 4 start.

PRXPAREN is used to determine whether we want the info for submatch 2 or 4. PRXPAREN returns the last submatch that was matched. If an area code of the format (XXX)XXX-XXXX is matches, PRXPAREN returns 2. If the format XXX-XXX-XXXX is matched, PRXPAREN returns 4.

```

data _null_;
  retain re areacode_re;
  length first last home business $ 16;
  length areacode $ 3;

  if _N_ = 1 then do;
    /* (XXX) XXX-XXXX */
    paren = "\\([2-9]\\d\\d\\) ?[2-9]\\d\\d-\\d\\d\\d\\d";

    /* XXX-XXX-XXXX */
    dash = "[2-9]\\d\\d-[2-9]\\d\\d-\\d\\d\\d\\d";

    /* Combine two phone patterns into one with a | */
    regexp = "/"( " || paren || " )|( " || dash || " )/";

    re = prxparse(regexp);
    if missing(re) then do;
      putlog "ERROR: Invalid regexp " regexp;
      stop;
    end;

    areacode_re = prxparse("/828|336|704|910|919|252/");
    if missing(areacode_re) then do;
      putlog "ERROR: Invalid area code regexp";
      stop;
    end;
  end;

  input first last home business;

  if ^prxmatch(re, home) then
    putlog "NOTE: Invalid home phone number for " first last home;

  if prxmatch(re, business) then do;
    which_format = prxparen(re);
    call prxposn(re, which_format, pos, len);
    areacode = substr(business, pos, len);
    if prxmatch(areacode_re, areacode) then
      put "In North Carolina: " first last business;
  end;
  else
    putlog "NOTE: Invalid business phone number for " first last business;
  datalines;
  Jerome Johnson (919)319-1677 (919)846-2198
  Romeo Montague 800-899-2164 360-973-6201
  Imani Rashid (508)852-2146 (508)366-9821

```

Palinor Kent 704-782-4673 704-782-3199
Ruby Archuleta 905-384-2839 905-328-3892
Takei Ito 704-298-2145 704-298-4738
Tom Joad 515-372-4829 515-389-2838
;

Output:

In North Carolina: Jerome Johnson (919)846-2198
In North Carolina: Palinor Kent 704-782-3199
In North Carolina: Takei Ito 704-298-4738

References

The metacharacter table in the Syntax section was taken from Microsoft's [web site](#).

The Complete Guide to SAS Indexes

By: Michael Raithel

List price: 54.95 USD

344 pages

ISBN: 1-59047-849-5

Publisher: SAS Press

Publication Date: January 2006

Description:

Your definitive guide to using SAS® indexes Do you struggle with the problems of extracting data from large SAS data sets? Are you designing applications where response time is of great importance? Let this example-driven book guide you to the many ways you can use SAS indexes to reduce the computer resources needed to process large SAS data sets, leading to faster programs and reduced computer charges. You will learn:

- The types of applications that can benefit from the use of indexes.
- How indexes can save computer processing resources and wallclock time.
- The difference between simple and composite SAS indexes.
- How to create indexes with the DATA step and with SAS procedures.
- The four program constructs that will get SAS to use your indexes.
- What centiles are and how you can use them.
- How to recover missing indexes and repair damaged indexes.
- And much more!

Whether you're a novice SAS user seeking to understand SAS indexes, an advanced user seeking a reference, or a SAS programmer working with large data sets, this book was written for you!

SAS Products Addressed: Base SAS

Releases: 9.1.3

Operating Systems: 64-bit Enabled AIX, 64-bit Enabled HP-UX, 64-bit Enabled Solaris, HP-UX IPF, Linux, Linux on Itanium, Microsoft Windows for IPF, OpenVMS Alpha, Tru64 UNIX, Windows, Windows NT Workstation, z/OS

Order it today! <http://www.sas.com/apps/pubscat/bookdetails.jsp?&pc=60409>

SAS and Google Partner on BI Search Capabilities

Just about everyone above the age of 5 knows how to "[Google](#)" -- but in most organizations, only a handful of power users know how to quickly locate data from high-tech business intelligence software.

That's all about to change. Beginning this summer, Google and SAS will offer mutual customers the familiar Google search interface to connect business users directly to the [SAS® Enterprise Intelligence Platform](#). Users of all skill levels will be able to search SAS and get real-time information, including data, analyses and reports, directly in Google search results pages.

Easy access to vital information

In the same way that SAS has expanded access to business intelligence (BI) by creating targeted user interfaces for its software that match the skill levels of individual users, SAS and Google will provide joint customers who activate the new Google OneBox for Enterprise feature of the Google Search Appliance with a familiar, secure way to search for real-time information delivered by SAS BI software.

Google OneBox for Enterprise uses the same technology that provides information on stock tickers or weather information on Google.com. Anyone who can perform a Google search can easily find the information they need, when they need it.

In addition, the combination of the Google Search Appliance with the SAS Enterprise Intelligence Platform will give users more information than ordinary keyword searches can provide. SAS' contextually relevant search capabilities with Google not only explore metadata, but also look at the business views (SAS Information Maps) that have been defined by SAS clients.

This means that search results will include data, analyses and reports that contain common information pertaining to the search phrase, giving users a broader and more relevant view of the search topic. A typical keyword search would return only reports that contained the keyword in the title, missing other key information pertaining to query. This information, typically untapped, is critical in supporting business decisions.

How it works: the IT perspective

Once a search term or phrase is entered, triggers (defined by the client) point to the SAS Metadata Server, included with many [SAS@9](#) offerings. In order to link together information based on underlying related data, SAS' contextually relevant search capabilities also search the business views/data structure determined by the client.

Users throughout an organization can easily view data, analyses and reports without involving IT to help locate and distribute the information -- and can access only information that they are already authorized to view.

How it works: the business perspective

Google OneBox for Enterprise, combined with SAS' contextually relevant search capabilities, delivers relevant search results for business users in the same way it does for

any visitor to the Google Web site looking for information about local news, weather or restaurants.

For example, when a SAS customer types a phrase such as "fourth quarter 2005 sales" into a Google-powered intranet search engine, it will return a snapshot of relevant information including reports, data and analyses along with links to other results -- top-selling products, top salespersons or top 10 customers for that time period, for example - data that would not be available through traditional keyword searches.

All search results are filtered through existing enterprise security protocols, delivering intelligence tailored to each user's individual access rights.

A big step in the right direction

"Google is excited to work with SAS to deliver business intelligence information and help employees spot trends right from their Google search box," says Dave Girouard, Vice President and General Manager of Google Enterprise. "We're aiming to make enterprise search as comprehensive and useful as Web search, and our partnership with SAS is a big step forward in that direction."

"SAS is committed to delivering the best business intelligence in the industry by enabling organizations to get access to relevant information when they need it," says Keith Collins, Senior Vice President and Chief Technology Officer of SAS. "The combination of SAS' business intelligence and Google's search expertise allows customers to quickly access and understand critical information by exposing associated data, analysis and reports to even more business users, broadening the value and impact of business intelligence and helping companies improve their own enterprise BI strategies."

This is the first of many technology initiatives that SAS and Google will spearhead to help organizations eliminate information silos by sharing relevant knowledge across business units.

New and Updated Guides for SAS® 9.1.3 Intelligence Platform

Business Intelligence guides offer a quick overview of the architecture behind the SAS Intelligence Platform, simple installation instructions, and the best way to administer your enterprise system. In this release, you get a new guide for performing security-related administrative tasks and a separate document of pre-installation checklists. The new and updated guides are available in SAS OnlineDoc (<http://support.sas.com/documentation/onlinedoc/sas9doc.html>) and on <http://support.sas.com/pubs>.

Analytics Training

As the undisputed leader in analytics knowledge and technology transfer, SAS offers comprehensive training that will help you reduce uncertainty, predict with precision, optimize performance and create value for your organization.

Discover all that SAS Analytics Training has to offer, including suggested course roadmaps, events and other training activities, by job roles and methodologies.

Find out more at <http://support.sas.com/training/analytics>

Sample 588: List all files that are located in a specific directory

This macro will allow you to print all files or certain files with a particular extension. This macro accepts 2 parameters. The first parameter is the directory that contains the files. The second parameter is the extension of the files you want to search for. If you want to list all the files, then leave the second parameter blank, for example %drive(c:\).

```
%macro drive(dir,ext);

  %let filrf=mydir;

  /* Assigns the fileref of mydir to the directory and opens the directory */
  %let rc=%sysfunc(filename(filrf,&dir));
  %let did=%sysfunc(dopen(&filrf));

  /* Returns the number of members in the directory */
  %let memcnt=%sysfunc(dnum(&did));

  /* Loops through entire directory */
  %do i = 1 %to &memcnt;

    /* Returns the extension from each file */
    %let name=%qscan(%qsysfunc(dread(&did,&i)),-1,.);

    /* Checks to see if file contains an extension */
    %if %quppercase(%qsysfunc(dread(&did,&i))) ne %quppercase(&ext) %then %do;

      /* Checks to see if the extension matches the parameter value */
      /* If condition is true prints the full name to the log */
      %if (%superq(ext) ne and %quppercase(&name) = %quppercase(&ext)) or
        (%superq(ext) = and %superq(name) ne) %then
        %put %qsysfunc(dread(&did,&i));
      %end;
    %end;

    /* Closes the directory */
    %let rc=%sysfunc(dclose(&did));

  %mend drive;

  /* First parameter is the directory of where your files are stored. */
  /* Second parameter is the extension you are looking for. */
  /* Leave 2nd parameter blank if you want a list of all the files. */
  %drive(c:\,sas)
```

SN-017298

How to create a user or system level environment variable outside the scope of the command or cmd session that it is created in

Environment variables created by a set command within a Command or CMD session are not available as user or system level environment variables outside the scope of the session that they are created in. Applications that run outside of the command or cmd session do not have access to environment variables created within the command or cmd session. Once the command or cmd session is exited, the environment variables created during the session no longer exist.

To address this issue Microsoft created a utility that can create global user and system environment variables from within a command or cmd session called SETX.

Information about downloading and using this utility is available:

http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/ntcmds_shelloverview.mspx

Product: Base SAS
Component: General System Issues
Priority: N/A
Note Type: Usage Issue
Date: Wed, 12 Apr 2006

Operating System and Source Fix Information

System	Release Reported	Release Fixed
Windows NT	9.1 TS1M0	
Windows 2000 Datacenter Server	9.1 TS1M0	
Windows 2000 Professional	9.1 TS1M0	
Windows 2000 Server and Advanced Server	9.1 TS1M0	
Windows 2003 Server and Advanced Server	9.1 TS1M0	
Windows XP	9.1 TS1M0	

Unless otherwise stated above, no fixes are available for this issue.

Webcasts and Events

Using SAS Software in the Design of Clinical Trials

Thursday, May 11

12:30-1:15 p.m. EST

Alex Dmitrienko, Ph.D., is Principal Research Scientist at Eli Lilly and Company. He is the coauthor of *Analysis of Clinical Trials Using SAS: A Practical Guide* and has published numerous papers.

Fundamental Forecasting Pains

Wednesday, May 17

1:30 p.m. ET

View this live Web seminar to learn more about the basic methods for measuring the elements of demand variation and for viewing the relationship between volatility and forecast accuracy.

F2006

June 5-6

Cary, NC

Learn the latest forecasting theories, trends and best practices from world-renowned forecasting experts at F2006.

JMP® User Conference

June 20-21

Cary, NC

Attend exciting and insightful sessions, plus roundtable discussions, a Scripting workshop, a Genomics Discovery event and exclusive training courses.

Inside SAS: Spotlight on SAS Enterprise BI Server

On-Demand Web Seminar

See a preview of the latest release of SAS Enterprise BI Server, including load balancing to maximize performance in heavy Web application environments and much more.

IT Management Summit on BI

Four-City live event

This series was created for busy IT professionals to acquire knowledge and insight from industry pundits as well as real-world experience on a variety of important topics concerning IT. Check the Web for dates and cities.