DBMS/COPY

The tool for software connectivity

Data Conversion
Data Analysis
Data Exploration

DBMS/COPY
DBMS/Analyst
DBMS/Explorer

Version 7
DBMS/COPY Version 7.0

September 3, 1999

All rights reserved.

Printed in the United States of America

No part of this publication may be reproduced, stored in a retrieval system, transmitted, or distributed, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of Conceptual Software, Inc.

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at 52.227-7013.
Contractor/manufacturer is: Conceptual Software, Inc.
9660 Hillcroft #510, Houston, TX 77096.
http://www.conceptual.com/

DBMS/COPY, DBMS/Analyst and DBMS/Explorer are trademarks of Conceptual Software, Inc.
# Contents

DBMS/COPY™ -- Quick Start ................................................................. 1

**General Information**

- Introduction to DBMS/COPY .......................................................... 5
- DBMS/COPY and Year 2000 ........................................................... 10
- New for DBMS/COPY Version 7 .................................................... 12
- DBMS/COPY Installation ............................................................... 14
- How DBMS/COPY Works .............................................................. 15
- DBMS/COPY - Problem Solving .................................................... 24

**Interactive Interface**

- DBMS/COPY - Interactive Interface ............................................ 26
- Reading Multiple Member Files .................................................. 30
- Reading Spreadsheets ............................................................... 31
- Reading ASCII Files ................................................................. 36
- Reading Microsoft Access .......................................................... 45
- Reading ODBC .......................................................................... 47
- Reading Value Labels ............................................................... 50
- The Power Panel ........................................................................ 53
  - Database Viewer ..................................................................... 54
  - Variable Information ............................................................ 55
  - Equation Builder .................................................................... 58
  - Sort ...................................................................................... 65
  - Data Information ..................................................................... 66
- Interactive Options ....................................................................... 69
- Writing Multiple Member Files .................................................... 71
- Writing ASCII files ..................................................................... 72
- Writing Microsoft Access Tables ................................................ 75
- Writing ODBC Tables ............................................................... 76
- Transfer Verification ................................................................. 77
- DBMS/COPY Interactive — The Command Line ......................... 78

**Batch Interface -- DBMS/COPY Plus**

- Creating Batch Programs ........................................................... 82
- Batch Editor ............................................................................... 85
- Running Batch Programs from the Command Line .................... 90
- Log File ..................................................................................... 92
DBMS/COPY Plus Modules

Overview ................................................................. 94
Syntax ........................................................................... 95
Error Messages .............................................................. 98
Compute Module .......................................................... 100
Contents Module ......................................................... 106
Datainfo Module .......................................................... 108
Sort Module ................................................................. 111

Module Reference

General Statements .......................................................... 113
Assignment Statement ....................................................... 115
Createformatvars Statement ............................................. 118
Database Specification Statement ................................. 120
Delete Statement ............................................................. 126
Drop Statement ............................................................... 128
Format Statement ............................................................ 130
f / Then Statement ........................................................ 136
Inodbc Statement ............................................................ 139
Keep Statement ............................................................... 141
Label Statement ............................................................. 143
Modify Statement ........................................................... 144
Obs Statement ................................................................. 146
Order Statement ............................................................. 147
Outodbc Statement ........................................................ 149
Rename Statement ........................................................... 151
Retain Statement ............................................................ 153
Run Statement ................................................................. 156
Select Statement ............................................................. 157
Stop Statement ............................................................... 159
Values Statement ............................................................. 160
Vartype Statement .......................................................... 161
Expressions ................................................................. 164
Functions ........................................................................ 178
   Numeric Function Listing ........................................... 184
   Character Functions .................................................. 206
   Miscellaneous Functions ........................................... 210

Included Program Overviews

DBMS/Explorer ............................................................. 212
DBMS/Analyst ............................................................... 229
Preface

Conceptual Software is pleased to announce that DBMS/COPY V7.0 is available for both Windows 95/98/NT and UNIX.

We are very excited about this release. For over 10 years, DBMS/COPY has been the foremost program for data transfer between packages. In celebration of the new millennium, and the second decade of DBMS/COPY, we are including a free license to DBMS/Analyst and DBMS/Explorer. These two programs push DBMS/COPY into the world of sophisticated multiple database manipulation and data exploration.

Please register your license!

Never before has registering your DBMS/COPY been so important. We are exploiting the power of the Internet to keep your licensed copy of DBMS/COPY up to date. When you register on our private e-mail list, we will inform you of new maintenance releases which can be downloaded from our web site. The help menu in DBMS/COPY V7 has a “check for update” menu item which will verify if your license to DBMS/COPY is up to date.
We wish to thank our many beta testers for spending countless hours helping to improve DBMS/COPY.

Dr. Fred Wolfe - for his efforts in helping us improve the speed of Microsoft Access and ODBC support. Bill Raynor - for supplying me with SAS for Unix files while I waited for my Sun computer to get updated. The fine folks at the Federal Bureau of Prisons (Yesvy, Nick and Bo) and Diana Fischer at Yale - for convincing me to move PRODAS to Windows. Robert Gerzoff - for his willingness to read and critique my manuals and his help in dismantling the CSI booth at the annual statistics conference. A special thank you goes to Jerry Dallal for testing DBMS/COPY and DBMS/Explorer.

Once again, I want to thank all of the beta testers for their invaluable assistance.

Last, but definitely not least, we especially want to thank our faithful DBMS/COPY users. We are proud that you have made DBMS/COPY your data transfer program of choice. Thank you for your support.
DBMS/COPY™ -- Quick Start

Welcome! So, you’re sitting at your desk and you have this dataset formatted for one software package but you want it formatted for another package. You have work to do in that other package and you want DBMS/COPY to transfer the data quickly and efficiently. You are very busy and would love to learn about the power of DBMS/COPY but right now, you just want to move the data. Am I right? These few pages will step you through a simple transfer. Just run the program, click on the Interactive button and do what makes sense. It’s that simple. You can always click the Help button or press F1 for help.

This section of the manual shows you how easy it is to transfer data. Detailed instructions are in the chapter, Using DBMS/COPY Interactive.

If you have not yet installed DBMS/COPY, do so now. Just run setup.exe located on the distribution media. Follow the on screen instructions. See the DBMS/COPY Installation page if you have any questions about installing.

In the example described below, the dbase file, saledbf.dbf will be transferred to a SAS for Windows file. The saledbf.dbf file is provided in the files subdirectory within the DBMS/COPY directory.

❖ Get The Ball Rolling

DBMS/COPY Interactive is started by clicking on the initial screen’s Interactive button:
Specify the Input Database

The input database dialog box is used to specify the input database. Scroll the **Files of Type** drop-down box to the desired input package. For this example, select the dBase file type. Navigate through the directory structure to locate your files. After you find the directory, select the file by double clicking on the name or single clicking the name and clicking the Open button.

The Power Panel

After the input database is selected, the **Power Panel** is displayed:
The Power Panel gives you total control over the transfer. From here you can select records, compute new variables, view the database, rename variables, and select variables. But for now, let’s move on to the output database. Just click Ok.

❖ Specify The Output Database

Now, the Select Output Database dialog box is displayed:

If the current output database type is not the desired one, just change it by opening the Files of Type drop-down box and select from the list of types displayed. In this example, SAS for Windows is selected as the output format: The default output name is the same name as the input package but with the proper extension.

Once you have the package you want, the directory you want and the filename you want, click the Save button to indicate you are done with the output database dialog box.
Quick Start

Transfer Verification

DBMS/COPY Interactive now shows you the batch equivalent of your interactive transfer. You can save the program for future execution. DBMS/COPY gives you the best of both worlds: Powerful interactive processing and automated batch processing. More about this later. To complete the conversion, click the Do-It! button.

Do-it!

DBMS/COPY displays a window while transferring the records. You will see record and variable counts. Here is the screen 75% done.

Once you get to 100%, you’re all done! You now have a copy of the dBase file formatted as a SAS for Windows file! The next section of the manual provides general information about DBMS/COPY. After that section, the DBMS/COPY interface is described in detail. Just remember that we designed DBMS/COPY to be both easy to use and powerful. Help is always available by hitting the F1 key.
Introduction to DBMS/COPY

Software Connectivity for Windows

DBMS/COPY is a utility that translates and transfers data between over 80 different database management systems, spreadsheets, statistical analysis, Microsoft Access, ODBC and other application packages while giving the user the ability to customize the output data file. For anyone who uses more than one database management system, or requires access to data stored in more than one system, DBMS/COPY offers a flexible and intelligent solution.

DBMS/COPY has two interfaces:

• DBMS/COPY Interactive
• DBMS/COPY Plus

Use DBMS/COPY Interactive for easy transfers. You are automatically prompted for the input database and the output database names. The Power Panel enables you to easily select variables, rename variables, compute new variables, select records, view the database, sort the data and get information on the data. DBMS/COPY Interactive can even save your work in a batch program file for use with DBMS/COPY Plus.

DBMS/COPY Plus creates and executes batch programs for repetitive transfers. The batch programs can also select variables, rename variables, compute new variables and select records. The batch program files can have any number of transfers. If you need to move 50 databases every Friday, this is the method.

DBMS/COPY Plus has been enhanced with the inclusion of a complementary license to DBMS/Analyst. DBMS/Analyst enhances the power of DBMS/COPY with a powerful multiple database manipulation and programming system. DBMS/Analyst helps clean and scrub data. DBMS/Analyst is briefly described later in this manual. The complete manual is included on-line.

Want to execute a transfer from another application or create an icon to start the transfer? No problem, DBMS/COPY senses if the input and output database names are on the Window’s command line. Also, batch program files can be executed from the Window’s command line.
The DBMS/COPY Plus Batch Editor combines a full window editor with built-in help and access to the DBMS/COPY Interactive Input and Output Database Dialog Boxes.

Overall, DBMS/COPY does an excellent job of transferring the data. There are limitations imposed by the input and output file systems, as well as the operating system. For example, DBMS/COPY can read from and write to most of the file structures. There are a few, however, for which DBMS/COPY only works in one direction. The limitations are discussed in the chapter How DBMS/COPY Works. The on-line help system documents the individual supported packages.

A very important feature of DBMS/COPY is that using DBMS/COPY will not change the input file in any way.

Register

Your purchase of DBMS/COPY entitles you to technical support. In order for us to provide the best possible service, this support must be restricted to registered users. You can register on the web (http://www.conceptual.com/) or mail the registration form to safeguard your rights as a DBMS/COPY owner. Registration also allows you to receive news of software updates and new applications.

You should register even if you are a registered user of a previous version.

Web Site

The Conceptual Software web site is http://www.conceptual.com/. The web site has a technical section including a knowledge base, frequently asked questions and technical product news. Please check it regularly. You can get to the web site from inside the program. The help menu has several web site jumps including one for maintenance updates.
DBMS/COPY Documentation Overview

- **DBMS/COPY General Information**

  DBMS/COPY - Introduction
  This chapter.

  DBMS/COPY and Year 2000 Compliance
  Information on how the year 2000 affects DBMS/COPY.

  DBMS/COPY Enhancements from V6
  What has been added to DBMS/COPY V7.

  DBMS/COPY Installation
  Please read this chapter before you install DBMS/COPY.
  If DBMS/COPY has already been installed on your system, you can skip this chapter.

  How DBMS/COPY Works
  The principles behind DBMS/COPY, and how the program converts numbers, dates, formats and variable names.

  Problem Solving for DBMS/COPY
  What to do if the file transfer does not work as expected.

- **DBMS/COPY User Interfaces**

  Using DBMS/COPY Interactive
  Information included in this chapter answers any questions you might have. Sample files and examples are provided for demonstration purposes.

  Using DBMS/COPY Interactive – The Command Line
  Simple transfers can be specified on the command line and even turned into an icon. Again, sample databases and examples are provided for demonstration purposes.

  Using DBMS/COPY Plus – The Batch Editor
  This chapter explains how the DBMS/COPY Plus interface works, and guides you through several examples.
Using DBMS/COPY Plus – The Command Line
How to run a batch file from the command line. How a command can be converted into an icon.

Using DBMS/COPY - Log File
DBMS/COPY maintains a complete log of all translations you have executed. See this section for an explanation of the information written to the log.

❖ DBMS/COPY Modules

Syntax
Overview of the required syntax for DBMS/COPY batch programs.

Compute Module
Within batch programs, the Compute Module allows users to access the DBMS/COPY record filtering, expression processing, and function library while transferring records from the input database to the output database. This chapter describes the proper syntax and commands to use with the compute module. For DBMS/COPY, this is used to create batch programs for repetitive executions.

Contents Module
The Contents Module lists the variables in a database.

Datainfo Module
The Datainfo Module generates a database of information about your database.

Sort Module
The Sort Module creates a sorted output database.
Compute Module Reference

General Statements  Statements that can be used in the Compute Module.

Expressions  Describes what can be included in an expression. Expressions are used in many of the general statements.

Functions  Functions can be included in general statements and expressions. Available functions range from simple math functions such as the absolute value function to complex financial and probability functions such as internal rate of return or gamma distribution.

Supported Packages

Each software system has its own characteristics and idiosyncrasies. DBMS/COPY has been designed to take these into account. The on-line help system describes each package individually. These help pages can be reached from the Help button on the Input and Output Dialog boxes.

Error Messages

Error message text can be found on-line.

Last Minute Notes

The DBMS/COPY software includes a file called README.TXT. This file contains information not available when this manual was printed. Please read the file for additional information.
The whole world seems to be going nuts about the year 2000. (Should we all schedule a 2 week vacation starting January 1, 2000?)

The good news is that most packages store dates in either julian dates (number of dates since some date in the past) or store 4 digit years.

Most of the problems will be with ascii files and dates that were entered as text strings containing two digits.

We have gone through the entire DBMS/COPY system including each and every database driver to resolve any year 2000 problems.

**DBMS/COPY Year 2000 Notes**

Year function
    Always returns a 4 digit year.

Julian and mdy functions
    If the year parameter is less than 100 then the century base will be the current century.

Year2digit function
    Always returns a 2 digit year.

2 digit years in constants are for the current century.
Year 2000 Tools

The Options dialog box under the Interactive’s menu lets you set a few year 2000 parameters.

Output Dates Always use 4 digits
Some software packages do not maintain a “date” variable type. For those packages, DBMS/COPY will write the date as a character string.

If this box is not checked, the dates will have 2 digit years if the date is in the current century and will have 4 digits if not.

If this box is checked, the dates will always have 4 digits.

Input Database 2 digit year start
A few input packages only store the year in two digits. With the new century coming, the question becomes what does a year 02 and a year 98 mean? This option lets you define the base for the 100 year range for what a two year means.

For example, if the base is 1950. The following examples will make this clear.

- Year 02 -> 2002
- Year 98 -> 1998
- Year 50 -> 1950
- Year 49 -> 2049

We feel that between the changes to the internals and the year 2000 tools, DBMS/COPY is well positioned for Jan 1, 2000.
New for DBMS/COPY Version 7

DBMS/COPY for Windows Version 7 is a 32-bit version for use with Windows 95/98/NT. If you are running Windows 3.1, you can order DBMS/COPY V6.

New For DBMS/COPY Version 7

We have added new supported packages, enhanced the support of existing packages, added new functions, new modules, and generally improved the usability of DBMS/COPY.

General Enhancements

Spreadsheet Grabber
Manual entering of the data range and variable name rows. Of course you can still use the mouse.

Microsoft Access Direct Support
Access is now on the list of packages. Just select the Access File type, find your Access database and open it. DBMS/COPY can now also create Access databases on output and overwrite existing tables.

ODBC Speed Faster
The ODBC support for SQL packages (Oracle, Sybase, Informix, DB2, for example) is significantly faster. We have seen speed increases of factors of 8 to 40 times. One user has a 100 times increase in speed.

ODBC Table Overwrite
Previously, DBMS/COPY only wrote to new tablenames. Now it can overwrite existing tables.

On-line Documentation
The entire DBMS/COPY, DBMS/Analyst and DBMS/Explorer manual set is now available as part of the help system. The help system uses the 32-bit version of the Windows help system.

Value Labels
This was experimental in Version 6 and now included in Version 7. Values labels from SAS, SPSS and Stata can be
read and written. When you move a dataset with value labels to a package which doesn't support value labels, you have DBMS/COPY create new variables containing the formatted values.

ASCII Data Dictionary Builder
   For free format files, DBMS/COPY will scan the ASCII file to determine the variable types and maximum lengths.

Web Maintenance
   Maintenance releases will be available on the Conceptual Software web site. New in Version 7, is user controlled checking for maintenance releases. There is a help menu entry for checking to see if your license is up to date.

New and Enhanced Packages

Excel       Read and write Excel 2000.

SAS         Up to Version 8. Read and write SAS for Unix included now included at no additional charge. Read SAS for VAX/VMS at no additional charge.

SAS Value Labels
   We had an experimental release of support of SAS Value labels in DBMS/COPY V6, now it is official.

SPSS        Unix and Macintosh support

Quattro Pro  Through Version 8 (.wb3)

Access      On the list of file types

Paradox     Up to version 9

Stata       Up to version 6

Systat      Up to version 9
DBMS/COPY Installation

DBMS/COPY Version 7 for Windows requires Windows 95/98/NT.

Installing DBMS/COPY for Windows

You received DBMS/COPY on either a CD-ROM or a set of floppies. You will find a setup program on the CD-ROM or floppy disk #1.

A series of instructions will lead you through the necessary steps.

Your first execution of DBMS/COPY will ask you for the serial number and give you an opportunity to register online.
How DBMS/COPY Works

DBMS/COPY really consists of two parts: DBMS/COPY Interactive reads data from any supported package and translates it to the format required by any supported output package. DBMS/COPY Plus creates and runs batch program files. DBMS/COPY employs a simple principle to perform a complex task.

Just Like Federal Express

Overnight delivery of packages between urban centers is taken for granted today, and a dozen major couriers struggle for shares of a highly competitive market. Sometimes forgotten is the radical innovation by which the best known courier, Federal Express, almost single-handedly brought the industry into being.

To all appearances, Federal Express delivers parcels between, for example, San Francisco and Los Angeles, between New York and Seattle, between Atlanta and Houston. What Federal Express actually does is considerably simpler: they transport every package, regardless of its ultimate destination, to the company hub in Memphis. Only in Memphis is it necessary to sort the packages, identify their destinations and assign them places on Federal Express aircraft bound for the target cities. In other words, all Federal Express really does is to transport parcels from disparate origins to a common destination, and then to transport the same parcels from a common origin to disparate destinations. The key to the system lies in breaking down a complex problem, moving packages quickly between scores of cities, into two elementary components joined at Memphis.

DBMS/COPY can't move records directly from SAS for Windows to Excel, from dBase to SPSS or from Dataease to Lotus any more than Federal Express can move parcels directly between Houston and New York. What DBMS/COPY does do is to read records written by these software systems, and copy the essential elements of those records into a specific place in memory. This is, for the purposes of our metaphor, analogous to flying a parcel to Memphis. Once in ‘Memphis,’ all records are functionally equivalent, regardless of the system under which they were originally created. Just as in the real Federal Express operations center, it does not matter whether a parcel was originally picked up in Portland or Peoria. DBMS/COPY then reads the record from memory and writes it to the destination system, just as the air courier flies parcels from
Memphis to Washington, to New York, to Chicago. By *pit-stopping* a record in a general form in memory, **DBMS/COPY** can read and then write any supported system to any other supported database system.

### Making Pit-Stops Work

Different systems take some very different approaches to the problem of reading, writing and organizing records. Under dBase III, numbers are stored as character strings with fixed decimal points. While in Paradox, the same data will be stored as binary floating point numbers. **DBMS/COPY** supports a large set of field types so it able handle the differences between packages.

Generally speaking, there are five potential problem spots **DBMS/COPY** confronts as it converts data between one system and another:

- File Extensions
- Variable types
- Variable value formats
- Missing values
- Variable names

The following pages provide a general description of how **DBMS/COPY** treats each of these potential problems. In addition, the information in the on-line help system includes specific information on how **DBMS/COPY** interacts with each package. There are greater details provided on pseudo extensions, variable types and formats, and idiosyncrasies of the particular package.

### File Extensions

Whether you use the Interactive system, the command line, or the Compute Module, the information sent to **DBMS/COPY** is essentially the same. The input (or source) and output (or destination) files and database systems are identified. The file names with their associated directories are just as expected under the operating system. The file extensions, however, may be different.

**DBMS/COPY** uses the file extensions to identify the software system for the file. For some systems with unique file extensions, the extension in the **DBMS/COPY** command can be the actual file extension. Examples of such systems include ABstat (.ab6) and Lotus 1-2-3, Version 2 (.wk1). Other systems may have file extensions in common. For example, seven of the systems that **DBMS/COPY** supports use .dat as a file extension.
and three use .sys. For these systems, the **DBMS/COPY** command uses a pseudo extension.

The pseudo extension replaces the disk file extension. The pseudo extension is only used by **DBMS/COPY**, never by Windows. It tells **DBMS/COPY** exactly what software package is associated with that particular file. Each package's on-line help, documents the actual disk extensions used by the packages, and the pseudo extensions used by **DBMS/COPY**.

Some database management systems are very similar to each other. **DBMS/COPY** uses the same pseudo extensions for these packages. For example, FoxBASE+, dBaseIII, and Clipper all use the .dbf extension.

**Variable Types**

Once **DBMS/COPY** has identified the source package and file, it reads the file. **DBMS/COPY** translates all the data from the original variables into its own internal variable types.

Below is a glossary of the **DBMS/COPY** internal variable types. Every database processed by **DBMS/COPY** has its variable types either mapped to or, translated from one or more of the types in the following table:
### DBMS/COPY Variable Types

<table>
<thead>
<tr>
<th>Vartype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Character string of length specified by input file.</td>
</tr>
<tr>
<td>double</td>
<td>8-byte floating point binary number. Double numbers are precise to 15 digits.</td>
</tr>
<tr>
<td>float</td>
<td>4-byte floating point binary number. Float numbers are precise to 7 digits.</td>
</tr>
<tr>
<td>short</td>
<td>Numbers from -32768 to 32767 (fit in 2-byte binary integers).</td>
</tr>
<tr>
<td>unshort</td>
<td>Numbers from 0 to 65535 (fit in 2-byte unsigned integers).</td>
</tr>
<tr>
<td>long</td>
<td>Numbers from -2 billion to 2 billion (fit in 4-byte integers).</td>
</tr>
<tr>
<td>unlong</td>
<td>Numbers from 0 to 4 billion (fit in 4-byte unsigned integers).</td>
</tr>
<tr>
<td>logic</td>
<td>Yes/no</td>
</tr>
<tr>
<td>date</td>
<td>Stores a date value (no hours or minutes).</td>
</tr>
<tr>
<td>time</td>
<td>Stores the time of day.</td>
</tr>
<tr>
<td>datetime</td>
<td>Stores both date and time in a single variable.</td>
</tr>
<tr>
<td>fixed</td>
<td>Fixed-point number with an established width and number of decimal places.</td>
</tr>
<tr>
<td>dollar</td>
<td>Number stores dollars and cents.</td>
</tr>
<tr>
<td>byte</td>
<td>Number from -128 to 127 (stored in 1 byte).</td>
</tr>
<tr>
<td>unbyte</td>
<td>Number from 0 to 255 (stored in unsigned 1 byte).</td>
</tr>
<tr>
<td>sasnum</td>
<td>SAS stores floating point numbers using 3 to 8 bytes. This specialized format accommodates them.</td>
</tr>
<tr>
<td>vary</td>
<td>Character string of varying length.</td>
</tr>
<tr>
<td>raw</td>
<td>Raw binary data of a determined length.</td>
</tr>
<tr>
<td>rowid</td>
<td>Rowid of an SQL table.</td>
</tr>
<tr>
<td>varname</td>
<td>Variable name with a length equal to the longest name supported by the output database</td>
</tr>
</tbody>
</table>
Variable Formats

In addition to translating the original information into the proper variable types for the output system, DBMS/COPY can often translate the format as well.

When DBMS/COPY encounters a numeric variable, it checks to see if there is a variable format assigned to that variable. If there is, the format is translated into one of DBMS/COPY's internal variable formats. If the output system supports variable formats, the format is passed to the output file. See the Format Statement in the Compute Module Reference section for a description of the DBMS/COPY variable formats. The information on each supported package describes which DBMS/COPY formats are assigned to each of the package's variable formats.

Just as numeric variables can have formats, so can date variables. DBMS/COPY for Windows treats date variable formats in the same manner as it treats numeric variable formats. The only format DBMS/COPY for Windows associates with character variables is the length.

Missing Values

A missing value is employed to indicate that on a particular record or observation a value for a variable potentially exists but is not available or missing. Although some software systems do not support the concept of missing values, the more sophisticated systems do allow for the possibility of missing values. They advise the analytical routines that the information is not available, and that it should not be used in any computations.

DBMS/COPY recognizes missing values, and transfers them intact from the input to the output database, provided the output database supports the concept of missing values.

Variable Names

Different systems employ different conventions for determining valid characters for variable names. In most cases, characters invalid under a target system are converted to underscores ( _ ) when DBMS/COPY writes a file. Exceptions are noted in each package's on-line help.

Some systems allow longer variable names than others. Generally, DBMS/COPY will truncate any variable names that are longer than the
target system allows. However, should this result in duplicate variable names, **DBMS/COPY** will modify the output automatically to make certain that each variable within an output file bears a unique name. If, for example, the variables *mail_address* and *mail_address2* are written to a database system in which variable names are limited to a maximum of 8 characters, simple truncation would render both of these variables as *mail_add*. **DBMS/COPY** replaces the last character of a duplicate character name with sequential numbers beginning with 1. Accordingly, the two hypothetical variable names would be translated as *mail_add* and *mail_ad1*.

Whenever the program performs any modification of variable names, the fact will be reported to the user in the Log File.

**Spreadsheets**

Spreadsheets consist of rows and columns. The rows may be thought of as the records in a database system, and the columns as variables. Therefore, a spreadsheet may be treated essentially as a database, which is how **DBMS/COPY** is designed to regard it.

The spreadsheet format does not, however, impose on its data the structure which a more conventional database would impart. **DBMS/COPY** therefore requires the user to specify certain structured features before the spreadsheet is processed. Specifically the user must identify which part of the spreadsheet contains data and which contains the variable names. The procedures for providing this information are discussed in the **DBMS/COPY User Interfaces** chapter of the manual.

**Providing Variable Names in Spreadsheets**

When **DBMS/COPY** reads the row or rows that have been designated as the variable names of an input spreadsheet, it expects to find character strings in each column. These become the variable names. All characters, including embedded blanks, are permitted in the variable names. If **DBMS/COPY** encounters columns without a character string in the variable name rows, these columns are not read.

**Supplying Variable Characteristics in Spreadsheets**

**DBMS/COPY** scans the data area to determine each column’s format and variable type. If all the cells in the column are numeric, **DBMS/COPY** assigns a numeric variable type to the output data from that column. If all the cells contain data with a date format, then the output data will be
assigned a date variable type. Otherwise, the output will be assigned the char variable type. DBMS/COPY Interactive’s Spreadsheet Variable Type Override dialog box will let you force the variable’s type.

Numeric and Date Variables
For numeric and date variables, DBMS/COPY checks the cell format to determine the most plausible variable output format.

Character Value in the Cells
If DBMS/COPY encounters a character string in any of the cells it is examining to determine the variable type, the variable will be read as a char variable. This variable will be assigned the column width or the length of the longest string in the cells, whichever is larger.

Writing Spreadsheets
When DBMS/COPY creates a spreadsheet, the variable names are always in the first row, and the data begins in row 2.

Access, SQL and ODBC
With Version 7, Microsoft Access support is done directly through the list of supported packages. You don’t have to use the ODBC interface. DBMS/COPY supports Microsoft ODBC (Open DataBase Connectivity). With ODBC, DBMS/COPY for Windows can process sophisticated SQL select statements. DBMS/COPY Interactive has a built-in Query Builder to help you create the Select statements. New tables can be created, data can be appended to existing tables and tables can be replaced. DBMS/COPY Plus can process SQL Select statements within the Compute module. To use ODBC for a particular software package (Oracle, Sybase and etc.) you must have installed the ODBC driver supplied by the database vendor.

Customizing Output Files
DBMS/COPY performs its data translations on a record by record basis. That is, it reads in all the data for one record, translates it to the output data type, and writes it to the output database before reading in the next record. While each record is in the central processor, or "Memphis," the user can make other changes. For instance, the data in two variables can be concatenated, converted to uppercase and stored in a new
variable. Variables that are not needed in the output file can be
excluded. Records that do not meet certain criteria can be excluded. The
original input file is not changed.

Within **DBMS/COPY Interactive**, variable renaming, variable exclusion,
new variable computations and record filtering is controlled by the Power
Panel. The Equation Builder is used to select records and compute new
variables. The Variable Information Dialog Box controls which variables
are written, the output names and labels.

Within **DBMS/COPY Plus** variable renaming, variable exclusion, new
variable computations and record filtering are accomplished with
command statements. The syntax and all statements are described in
later chapters of this manual. The following examples are provided to
illustrate the principals being discussed, not the specific syntax required.
(All examples are syntactically correct.)

The user can make changes to the output file by using the **DBMS/COPY
Plus** statements in what is called a paragraph. Consider the following
example,

```
compute;
    in= employee.rxd out= employee.db;
    newsal = salary * 1.1;
    drop depart;
run;
```

This sample paragraph reads in a Reflex file called employee and writes a
Paradox file. In an assignment statement, a new variable, *newsal*, is the
amount of the salary (already stored in the *salary* variable in the input
file) after a 10% raise is calculated. The Paradox file will not contain the
variable called *depart*, because it is excluded in the Drop Statement.

Assignment Statements can be conditionally executed. For example, the
paragraph above could have been written to give raises only to employees
who have been employed for five or more years.

```
compute;
    in= employee.rxd out= employee.db;
    if years > 4 then
       newsal = salary * 1.1;
    else
       newsal = salary;
    drop depart;
run;
```
The Assignment Statement calculates the value of `newsal` one way if a certain condition is met (`years` is more than 4) and another way if the condition is not met.

In addition to these straightforward operations, you have access to a large function library to create new variable values. You can rename variables, change their length, among many possibilities.

Your free license to DBMS/Analyst extends the capabilities of DBMS/COPY Plus. You can manipulate more than one input database, split that data into multiple output databases, write reports, read complex text files, do array processing, sort, transpose data, tabulate data, generate summary statistics and calculate regressions. There’s also a powerful macro scripting language. A built-in macro easily converts multiple files from one format to another.

**Limitations**

The development of DBMS/COPY, required a detailed knowledge of the file structures used by the different software products. Many software companies were very helpful in providing that information, and we are grateful to them.

Other companies, however, either did not provide file structure information or provided incomplete or incorrect information. In these cases, the programming staff was forced to decipher the file structures by hand. The results have been good, but be aware that there may be occasional difficulties that arise as a result.
DBMS/COPY - Problem Solving

DBMS/COPY has been designed to work correctly in almost all circumstances. Sometimes, however, the transfer may not work as you expected.

If you encounter difficulties in using DBMS/COPY Interactive, please take the following steps, in order:

- Check the manual again. There may be some seemingly minor point that will resolve the problem.
- If you are using the command line, try the same transfer using the Interactive interface.

If you have problems in using DBMS/COPY Plus, please take the following steps, in order:

- Try the transfer using the DBMS/COPY Interactive system. This will establish whether the problem is related to the file transfer itself or to some other DBMS/COPY Plus command.
- See the on-line help for Error which lists most of the possible error messages along with suggestions for correcting the problems. This information may help you determine what went wrong and why.

If none of these steps work, please check for an update. We store the maintenance releases on our web site. You can check for a release from the Help menu’s "Check for Update" item. It will connect to our web site and determine if you have the latest version. If you don’t have internet access on the machine with DBMS/COPY, please go to the "Technical" page of our main web site (http://www.conceptual.com/). If all else fails please contact Conceptual Software, Inc. for technical support.
Technical Support

Conceptual Software, Inc. offers technical support to its registered users. If you need help, contact us at one of the numbers listed below. Before you call, please have the serial number of your copy of DBMS/COPY available. (It is displayed in the About Dialog Box.)

Please have a copy of the DBMS/COPY Plus paragraph you are using available when you call. If it is a long paragraph, it may be helpful to email or FAX a copy of it to us before you call.

Please note any error messages you received.

It is very helpful if you are at your computer when you call.

If you wish, you can email, fax or regular mail the problem files to us. Be sure to include a description of the problem, including the source and target software systems.

Conceptual Software, Inc.
9660 Hillcroft Suite 510
Houston, TX 77096

Voice: (713) 721-4200
Fax: (713) 721-4298
Web: www.conceptual.com
Internet: support@conceptual.com
DBMS/COPY - Interactive Interface

This section details the operation of the Interactive interface for DBMS/COPY. If you have not read the DBMS/COPY - Quick Start, please read it now. This section will build on the Quick Start.

The Interactive interface is divided into 4 parts:

- Input Database Selection
- Power Panel
- Output Database Selection
- Transfer Verification

Input Database Selection

For most packages, selecting the disk file name is all that is needed to specify the input database. Additional processing is needed for multiple member files, spreadsheets, ASCII files, Access, and ODBC. Each of these types is explained below.

Multiple Member

Some packages store multiple sets of data within one disk file. After selecting the disk file, you will see the list of members.

Spreadsheets

Data can be stored anywhere within a spreadsheet. DBMS/COPY’s Spreadsheet Grabber lets you visually specify the block of cells where the data is stored.

ASCII files

Both fixed format and free format text files can be processed. To properly interpret ASCII files, the user must provide a list of field names, field types and field lengths. DBMS/COPY’s ASCII Data Dictionary Builder helps you create the field list.

Access

DBMS/COPY Version 7 has special support for Microsoft Access because it is so popular. The ODBC driver specification is done automatically after the database is selected.

ODBC

Microsoft’s Open DataBase Connectivity routines let you access packages (like Oracle, Sybase and others) with a standardized SQL select statement
interface. DBMS/COPY's SQL Query Builder helps you create the select statement.

Value Labels

A few packages including SAS, SPSS and Stata support value labels. These are mappings of raw values like 1,2,3 to display values like Yes, No, Maybe. DBMS/COPY can pass the value labels between the packages and/or create new variables containing the formatted values.

Power Panel

The Power Panel gives the user the ability to turn a simple transfer into a sophisticated data transformation. The five buttons on the Power Panel Dialog Box provide access to three areas:

Equation Builder

In this window the user can create new variables and establish a record filter using a library of over 100 functions and sophisticated expression processing.

Database Viewer

in this window the user can view the database records.

Variable Information

This dialog box allows you to select variables, rename them, change their type, and assign labels.

Sort

Sort the output database.

Datainfo

Get general statistics, percentiles and variable values.
Output Database Specification

After specifying the input database and any needed modifications in the Power Panel the only item left is specifying where to write the data. For most packages selecting the disk file name is all that is needed to specify the output database. Additional processing is needed for multiple member files, ASCII files, Access, ODBC, and time series files.

Multiple Member  Some packages store multiple sets of data within one disk file. After selecting the disk file, a list of members will be shown. Either enter a new name or select a name to overwrite from the list.

ASCII files  Text files are written in either fixed or free format.

Access  DBMS/COPY Version 7 has special support for Microsoft Access because it is so popular. The ODBC driver specification is done automatically after the database is selected. Tables can be created, appended to or overwritten.

ODBC  Creates, appends or overwrites tables using the Microsoft Open DataBase Connectivity routines.

Time Series  Time series software packages need to be provided with the beginning date and time interval associated with the data. The dialog box shows the available intervals and prompts for the date associated with the first record written to the output database. No additional detailed documentation is needed.

Empty Variables  A few packages, including NCSS, need to have extra space stored on the output database if you want to create new variables at a later time. This dialog box provided you an opportunity to specify the number of extra variable spaces. No additional detailed documentation is needed.
Transfer Verification
DBMS/COPY provides two interfaces: the easy to use Interactive interface and the automated batch processing in the Plus interface. What if you want to save an Interactive transfer so you can run it automatically next time? The Transfer Verification dialog box shows you the equivalent batch code of the interactive transfer. With DBMS/COPY you get the best of both worlds: Interactive and Batch!

Executing the Transfer
When you want to execute the transfer, just click the Do-it! button.
Some software packages including for example, SAS Transport and DataEase, store multiple sets of data within one disk file.

The following dialog box allows you to select the member of the disk file to transfer.
DBMS/COPY Interactive - Reading Spreadsheets

The process of reading a spreadsheet is different because part of the spreadsheet contains data, part contains the variable names and part may contain other miscellaneous information. The variable names may be next to the data, or may not even be included. The DBMS/COPY Interactive system will allow you to specify the portion of the spreadsheet containing the data and the variable names so it can properly be translated.

Note: Spreadsheets with variable names on row 1 and data area starting on row 2 do not have to be defined. Just click the OK button! DBMS/COPY will prompt you to make sure that you understand that the assumption will be names on row 1 and data starting on row 2.

The next example translates the Excel Version 5 ar_v5.xls (accounts receivable) file.

After selecting the file with the Input Database Dialog Box, you will see the spreadsheet displayed in the SpreadSheet Grabber™ window.
The Spreadsheet Grabber™ displays the contents of the spreadsheet. Two buttons at the top of the window are labeled Data Area and Variable Names. The Data Area item is selected when the window opens. The data area can be defined using the mouse or manually typing in the range.

**Using the Mouse To Define The Data Area**

First, click the mouse in the upper left cell of the data area. That cell location will then be stored in the first Data Area text box. Next, either hold the mouse button down and drag it to the lower right corner or position the mouse on the lower right data cell and Shift-click. You can also use the scroll bars to get to the end of the spreadsheet. In either case the data area will be displayed in reverse video and the ending cell coordinates will be displayed in the second data area text box.

In the example spreadsheet, the data is contained in rows 9 to 18 and columns A to F. Click the mouse on cell A9 and then either Shift-click on F18 or hold the mouse button down and slide the mouse until you get to F18.
**Manually Defining The Data Area**

In the first text box next to "Data Area", enter the text **A9**. In this example, the ending point is just fine so no need to change it.

**Defining the Variable Name Rows**

After selecting the data area, click the **Variable Names** button.

This will cause the display to switch allowing you to specify the variable name rows. The data rows will be shown with a border, red background and italics. Identify the rows containing the variable names by either entering the top and bottom rows manually in the "Variable Name Rows" text blocks or by clicking the mouse on the top row of the names and dragging the mouse to the bottom row of the names. The variable name rows will be shown in bold. Variable names must be in the same columns as the data and must range across all data columns. As you move the mouse you will be defining the names for all data columns automatically.

The variable names for the example spreadsheet are in rows 7 and 8. Click the mouse on row 7 and then either shift-click on row 8 or drag the mouse down to row 8. As you select variable rows, the row text will show in bold. As shown in the following window:
After defining the data and variable names, click OK.

### Spreadsheet Variable Type Override

After you have selected the data area and variable name rows from the Spreadsheet Grabber, DBMS/COPY will scan the spreadsheet. DBMS/COPY will count the number of character cells, numeric cells, date cells, time cells and datetime cells which were found in each column. Also, DBMS/COPY will determine the name associated with the column.

Normally the rule is if DBMS/COPY finds a column with mixed results (numeric and character) then the column will become character. This makes sense because you can always represent a number in a character string but not the other way around. The problem is frequently people will enter things like "didn’t answer" or "missing" in a numeric column.

This dialog box gives you the information you need to make an intelligent decision about the variable types.

![DBMS/COPY Spreadsheet Variable Type Override](image)

Columns where more than one type of field were found will have the counts in **bold**. For example, in column E had 1 character value but 10 date values so we could change the type to Date.

Column descriptions are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Created name of the variable. This field can be modified.</td>
</tr>
</tbody>
</table>
Drop  Contains a check box for each variable. If checked the variable will be skipped.

Type  Displays the determined variable type. It could be character, numeric, date, time or datetime depending on the capabilities of the spreadsheet. This field can be modified.

Length  Displays the length character field.

# Char  Number of character cells in the column.

# Num  Number of numeric cells in the column.

# Date  Number of date cells in the column.

# Time  Number of time cells in the column.

# Datetime  Number of datetime cells in the column.

You might or might not have all of the count columns. It depends on the capabilities of the spreadsheet.

**Default Spreadsheet Structure**

Spreadsheets with variable names on row 1 and data area starting on row 2 do not have to be defined. Just click the OK button! DBMS/COPY will prompt you to make sure that you understand that the assumption will be names on row 1 and data starting on row 2.

**No Variable Names on the Spreadsheet**

If the spreadsheet doesn't have variable names, just define the data area and set the variable name rows to zero. DBMS/COPY will use the column names (A, B, C, ...) as the variable names.

**Multiple Page Spreadsheets**

The latest generation of spreadsheets allow the actual disk file to contain multiple pages. If DBMS/COPY determines that the spreadsheet is multipaged, you will see buttons: **Next Page** and **Prev Page**. These two buttons will step through the pages of the spreadsheet. All of the data for a transfer must come from one page. If you have multiple sheets each sheet must be transferred separately.
ASCII files are also known as text files.

Although ASCII files are human readable, there is no standard ASCII file structure. ASCII files may have a fixed or free format and lines may or may not end in carriage returns, to name just two of many possible file characteristics. Each software system imposes its own requirements on the format of the ASCII files it can read and write. DBMS/COPY has a large number of options that the user can set to read and write a wide variety of ASCII file structures.

**General Approach**

When you tell DBMS/COPY to translate a file from one supported software system to another, DBMS/COPY already knows the file structure of the source and the destination files. DBMS/COPY observes the formats and protocol of these packages.

ASCII is not a software system; nor is there a standard ASCII file structure. So, if the file you will be transferring to or from is an ASCII file, you must tell DBMS/COPY the file's structure. If the ASCII file is the source file, you will have to tell DBMS/COPY about the variables and their locations within the record. If the destination file is an ascii file, you will need to tell DBMS/COPY the structure of the output file you want it to create.

You will use the data dictionary file to describe an input ASCII file. It specifies the arrangement of the data in the ASCII file. It names the variables' positions, names, types and lengths. The file name must be the same as the ASCII data file name except the disk file extension must be .dct.

DBMS/COPY Interactive creates and maintains the data dictionary file. In the examples given below, the sample file INVENTOR.DAT has been selected as the input file. This ASCII file is located in the FILES subdirectory, within the DBMS/COPY directory.
Translating Data From an ASCII File

Two dialog boxes are used to specify the data dictionary for an input ASCII file. The two dialog boxes are:

**ASCII Input Format Options**
- Defines general characteristics of the input file.

**ASCII Data Dictionary**
- Defines each field including its position, length, type and name contained within the input file.

**ASCII Input Format Options Dialog Box**

After the ASCII input file is selected, **DBMS/COPY** presents the **ASCII Input Format Options** dialog box:

The top half of the dialog box shows the first several lines of the input file. The file shown, `inventor.dat`, is a fixed format file. Fixed format files have each field in the same location on each record.
The user’s first task is to identify for DBMS/COPY if the file is fixed or free format.

**File is Fixed Format**

The check box toggles between a display of options for fixed format files and a list of options for free format files. You should set the state of the check box before moving on to the other options. The dialog box shown above, shows the options for free format files.

**Fixed Format File Options**

When the *File is Fixed Format* box is checked, the bottom half of the dialog box will appear as follows:

```
* File Is Fixed Format

Date Value Format  mm/dd/yyyy
Numeric Missing Value  .
Lines Per Record  1

Ok  Cancel
```

**Shared Fixed & Free Format Options**

**Date Value Format**

This box displays a drop-down list of input date formats. When date variables are present within an ASCII file, the format selected here will determine how DBMS/COPY processes those values. The formats can be separated by special characters, such as the forward slash or hyphen. Embedded spaces are not permitted.

**Numeric Missing Value**

The value provided here identifies the string that DBMS/COPY will interpret as a numeric missing value. For example, some files store a -999 as a missing value.
**Fixed Format Option**

**Lines Per Record**
Specifies the number of lines on the ASCII file that represent one record. In most cases, the default of one line per record is appropriate.

**Free Format Options**

**Field Separator**
The field separator is the most critical information needed by DBMS/COPY. The character entered here is the delimiter character and tells **DBMS/COPY** how to break up the data into fields. It indicates where one field ends and another begins. Common delimiters are comma, space and tab. To use a space or comma, simply enter the key itself. To use a tab, simply type the word **tab**. The default is the comma.

**Surround Character**
In certain cases the delimiting character may be included as part of valid data within the file. If the space character is defined as the field delimiter, then a name consisting of a first name and a last name would be divided into two fields. To avoid this problem, the **Surround Character** option tells **DBMS/COPY** to ignore any delimiter contained within sets of the character defined here. Frequently a double quote is used.

**Variable Names on Row #1**
This check box tells DBMS/COPY that the first row of the input file contains variable names. If the data dictionary does not already exist, DBMS/COPY will use the names in first row as the initial variable names.
**Fixed Format Data Dictionary Builder**

After the general options are set, DBMS/COPY brings up the **ASCII Dictionary Builder** dialog box for specifying the variable names.

The purpose of the ASCII Dictionary Builder is to define the starting position, length, type and name of each field present within the ASCII input file. As before, the top half of the window displays the initial contents of the file. The lower half contains entry boxes used to define field parameters. The list box in the middle displays already created fields. For each field the following parameters can be defined:

- **Start Column** Starting position of a field.
- **Length** Length of field.
- **Type** Type of field. A drop-down list displays available formats.
- **Name** The variable name given to the field.

After the initial field is defined, the **Add** button is selected to insert the next field definition, and the process is repeated. Previously defined fields may be edited by highlighting the definition from the list and
adjusting the parameters displayed for the field. To delete a previously defined field, highlight the field in the list, then select the **Delete** button.

In the example above, the ASCII database, INVENTOR.DAT, was retrieved as the input file. When all fields for the input database have been defined, the resulting field parameters list would appear as:

<table>
<thead>
<tr>
<th>Col</th>
<th>Len</th>
<th>Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>Char</td>
<td>Year</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>Char</td>
<td>Invention</td>
</tr>
<tr>
<td>22</td>
<td>12</td>
<td>Char</td>
<td>Inventor</td>
</tr>
<tr>
<td>34</td>
<td>11</td>
<td>Char</td>
<td>Country</td>
</tr>
</tbody>
</table>

At the completion of the field definition procedure, the **Finished** button may be selected to retain all definitions. The **Cancel** button will exit the ASCII Dictionary Builder without saving changes, returning control to the **DBMS/COPY** desktop.

**Fixed Format Data Dictionary Structure**

The **DBMS/COPY Interactive** interface hides the internal structure of the data dictionary. But if you are curious, the following is the dictionary file `inventor.dct`. The dictionary file is an ASCII text file which you can edit or create elsewhere.

```plaintext
dictionary
extension=DAT
missing=.
numeric=n
fixed=y
dictionary=dct
date=mm/dd/yyyy
variables
  1      5 c Year
  6     16 c Invention
  22    12 c Inventor
  34    11 c Country
endvars
```

The information provided between the words **dictionary** and **variables** is general format information about the `inventor.dat` file. The data between the words **variables** and **endvars** provides specific information about the fields.
For detailed information concerning the contents of a dictionary file, see the ASCII package in the on-line help system from the Input Database Dialog Box.

**Free Format Data Dictionary Builder**

For free format files, the Data Dictionary Builder dialog box similar to the following will be shown:

The purpose of the ASCII Dictionary Builder is to define the type and name of fields in the file. As before, the top half of the window displays the first few rows of the file. The lower half contains entry boxes to define the type and name of each field. The list box in the middle displays the already created fields.

The order of the fields defines the order of variables in each record. The text from the beginning of the record to the first delimiter is the first field. The text between the first and second delimiter is the second field and so on. The following information is needed for each field:

**Length**

The number specified here defines the maximum length for a character variable. DBMS/COPY will always skip to
the next delimiter before processing the next field. For all other field types the entire text between the delimiters is used. Note: if you leave the width at zero, DBMS/COPY will scan the file and determine the maximum width.

**Type**
Defines the field type. A drop-down list displays available formats.

**Name**
The variable name given to the field.

Fields are modified by highlighting the field in the list and then changing the values in the length, type and name elements. The **Delete** button will only delete the last field. At the completion of the field definition procedure, the **Finished** button may be selected to retain all definitions. The **Cancel** button exits the ASCII Dictionary Builder without saving changes, and returns control to the DBMS/COPY desktop.

**Free Format Data Dictionary Structure**
The DBMS/COPY Interactive interface hides the internal structure of the data dictionary. But if you are curious, the following is the dictionary file `employee.dct`. The dictionary file is an ASCII text file which you can edit or create elsewhere.

```plaintext
dictionary
names=y
separator=,
mustsurround=n
surroundchar="
quotechar='
extension=dat
missing=.
umeric=n
fixed=n
dictionary=dct
date=mm/dd/yyyy
variables
  1     15  c depart
  17    1  c sex
  19    12  R salary
  32    20  c jobtitle
  53    12  R rating
  66    12  R years
  79    10  D review date
endvars
```
The information provided between the words *dictionary* and *variables* provides DBMS/COPY with general format information about the EMPLOYEE.DAT file. The data between the words *variables* and *endvars* provides specific information about the fields.

For detailed information concerning the contents of a dictionary file, see the on-line help for ASCII from the Input Database Dialog Box.
Microsoft Access databases have an extension of .MDB. Each Microsoft Access database can store many tables. With DBMS/COPY you can retrieve some or all of the fields for one table or create a complex multiple table join. This is all accomplished with our SQL select statement builder. (SQL is an abbreviation for _Structured Query Language_.)

After you have selected the Access database in the Input Dialog Box, you will see the following DBMS/COPY dialog box:

Within the Select Statement Builder, you can select the table or tables, select the variables, and create any generalized select statement.

You will see two lists in the dialog box:

- **Tables** list of the tables in the Access database
- **Variables** list of variables on the currently highlighted table. Below the Variable's list is a one line text block were information about the currently highlighted variable will be display.

**Select Statement**

The text block is for the creation of the select statement.
Operation

The automatic select statement builder options are:

Automatic Table Insert

For example, if you double click on a table (or highlight a table and click Paste) while the Select Statement text block is empty, the following text will automatically be added.

```
select * from tablename
```

Which is actually sufficient to select every variable from the table.

Automatic Variable Insert

For example, if you double click on a variable (or highlight a variable and click Paste) while the Select Statement text block is empty, the following text will automatically be added.

```
select variable from tablename
```

Which is sufficient to select one variable from the table. You can paste addition variables into that statement by just double clicking on the variables. (The cursor's insert point is constantly updated for this operation.)

Complex Select Statements

The select statement text block can contain as complex a statement as you want. You can join tables, specify a where clause, order by or other database specific parameters. Please see your Microsoft Access documentation for the proper syntax for complex queries.
Many sophisticated database systems (Oracle, Sybase, Informix, DB2 and others) support a data access method called **SQL**. SQL is an abbreviation for *Structured Query Language*. One problem with SQL is that every software company has a slightly different SQL syntax. Another problem is that each vendor distributes a unique set of routines that programs, like DBMS/COPY, use to access their data.

In an attempt to solve these two problems, Microsoft defined a standard for accessing data stored in SQL based systems. The standard is called **ODBC**. ODBC is an abbreviation for Open DataBase Connectivity. DBMS/COPY supports ODBC for reading and writing.

DBMS/COPY for Windows has a direct connection to the ODBC routines. DBMS/COPY supplies an easy to use Query Builder. You don’t have to know the ins and outs of the SQL select statement to access ODBC data!

### Starting ODBC

Clicking the ODBC SQL button in the Input Database dialog box starts the ODBC subsystem. The user’s first task within the ODBC subsystem is to connect and login in an **ODBC Source**. The Source is the name of the database package from which you want to read data. The source dialog box is populated with the list of installed ODBC drivers. The ODBC Source’s dialog box and the login dialog box are managed by the ODBC subsystem and not DBMS/COPY. The ODBC subsystem is on the Window’s Control Panel. DBMS/COPY does not ship ODBC drivers. The vendor of the database system you want to access should be able to supply you with the ODBC driver and installation program. Most vendors maintain up to date drivers on their web site.
After you have connected to the ODBC source you will see the following DBMS/COPY dialog box:

Within the Select Statement Builder, you can select the table or tables, select the variables, and basically create any generalized select statement you want.

You will see two lists in the dialog box:

**Tables** list of the tables in the database

**Variables** list of variables on the currently highlighted table.

Below the Variable's list is a one line text block were information about the currently highlighted variable will be display.

**Select Statement**

The text block is for the creation of the select statement.
**Operation**

The dialog box can do a few things automatically to help you build the select statement.

**Automatic Table Insert**

For example, if you double click on a table (or highlight a table and click Paste) while the Select Statement text block is empty, the following text will automatically be added.

```
select * from tablename
```

Which is actually sufficient to select every variable from the table.

**Automatic Variable Insert**

For example, if you double click on a variable (or highlight a variable and click Paste) while the Select Statement text block is empty, the following text will automatically be added.

```
select variable from tablename
```

Which is sufficient to select one variable from the table. You can paste addition variables into that statement by just double clicking on the variables. (The cursor's insert point is constantly updated for this operation.)

**Complex Select Statements**

The select statement text block can contain as complex a statement as you want. You can join tables, specify a where clause, order by or other database specific parameters.

Our select statement builder does not provide routines for complex statements because they are frequently database system specific. Please see your database system documentation to see what it can do.
DBMS/COPY Interactive - Reading Value Labels

A few packages including SAS, SPSS and Stata support value labels. Value labels map raw values like 1,2,3 to display values like Yes, No, Maybe. DBMS/COPY can pass the value labels between the packages and/or create new variables containing the formatted values.

SPSS, Stata and other Non-SAS packages storing value labels

SPSS and Stata, among others, store the value labels in the same data file as the raw data. When DBMS/COPY reads the data file it is also able to read value labels. The input value label list is always loaded and always passed to the output package. The one problem is that not many packages support value labels so if nothing is done, when the data is written to these output packages, the value labels are lost. To give you an opportunity to capture the value labels, DBMS/COPY will display the following dialog box when it reads a dataset containing value labels:

DBMS/COPY can create new variables that will contain the formatted values of each variable that has a value label list. The name of the new variable will be the name of the raw variable with a prefix and/or suffix. The new variable will be a character variable and the length will be sufficient to store the longest value label. If you don’t want to create new variables, just click Done without entering a prefix or suffix.

For example, if two variables, jobcode and salarycode have formats and if you enter f_ as a prefix then two new variables f_jobcode and f_salarycode will be created.

If you only want to capture the formatted values of a variable or two, you can use the valuelabel function. It returns the formatted value for a
specified variable. Please see the function documentation for more information.

If you are writing value labels to SAS, please read the paragraph at the end of this section on how output value labels to SAS are handled.

**SAS value lists**

SAS stores the value labels in special format library files. These have a .sc2 extension. DBMS/COPY can read these special files and retrieve the value labels. Unfortunately, the raw data file does not store a list of the format libraries that contain the necessary value lists. In the raw data file, variables that have a user defined format have a format name that doesn’t match one of the standard SAS format names (date, comma, dollar and so on.)

The following dialog box will be displayed when you read a SAS dataset which has unknown formats.

![SAS Format Finder](image)

**Unknown Formats**

list shows the user defined formats that were found on the input dataset.

**Previously Seen Format Libraries**

list shows the format files that have been previously used. The program assumes that you will frequently be using the same formats. This handy list lets you quickly reuse the format libraries.

**Use From List**

tells the finder to scan the highlighted format libraries.
**Add File...** will bring up an Open File dialog box so you can specify another format library. It will automatically be scanned.

As format libraries are scanned, the list of unknown formats will shrink. You can give up looking for unknown formats at any time. It is not necessary to select anything. You can skip the user defined formats by just clicking Done. Two operations can be done with the formats.

1. When going to a package that support value labels (SPSS and Stata for example), the values from the SAS formats will be used to automatically create value labels in the target file.

2. Optionally, new variables can be created which will hold the formatted valued for each record. Most packages do not support value labels. This feature enables you to get the raw value and the formatted value in the same file. You must specify a prefix and/or suffix string that will be attached to the variable name to generate a new variable name to hold the formatted value. For example, if the prefix is f_ and the variable is jobcode, the new variable that will hold the formatted value of the jobcode variable would be f_jobcode. The new variable will be of type character with a length equal to the longest display value.

**Writing Value Labels To SAS**

As much as we would like to create a format library for you, the structure of the .sc2 has proved too complex to create. Instead DBMS/COPY will generate the syntax for proc format that you can run as a SAS program. The program file will have the extension of .sasprg and will be in the same directory and have the same filename as output SAS dataset. DBMS/COPY will store the format id in the SAS file which corresponds to the format in the .sasprg file.
After selecting an input database the following Power Panel dialog box appear.

The **Power Panel** enables you to customize the transfer of data from the input database to the output database. The **Equation Builder** is used to define a record filter and compute new variables. The **Database Viewer** displays the input data. The **Variable Information** dialog box controls which variables are written to the output database, their names, labels and types. The **Sort** routine generates sorted data on output. **Data Information** creates a summary of your data including general statistics, percentiles and variable values. If you skip the Power Panel (as we did in the Quick Start) the output database will write every variable for every record.
Selecting the **View Data** button on the Power Panel causes DBMS/COPY to display the contents of the input database. The database window scrolls across to see more variables and down to see more records.

Even though we have yet to discuss the Equation Builder, please no note that the Database Viewer reflects the Equation Builder's record filtering and new variable computations.

The six vertical scroll bar buttons perform the following functions (in descending order down the scroll bar): First record, Up a page, up a record, down a record, down a page and last record.

Going to the last record in a large database can be time consuming. To close the window, hit the **Esc** key or double click on the upper left close button.
Selecting the **Variable Information** button on the Power Panel causes DBMS/COPY to display a list of the variables on the database. Note: the Variable Information dialog box will include any variables created in the equation builder. When changes are completed in the Variable Information window, control is returned to the Power Panel.

The **Variable Information** window provides the ability to change the variable types, adjust the field lengths and change the variable names. The columns displayed are explained below:

**Name**
Displays variable names as they exist in the input database or computed in the Equation Builder. The variable list is displayed in alphabetical order.

**Rename**
New variable names for existing variables may be written here. The new variable names will appear in the output database.

**Position**
Displays the position of variables in the input database.

**Drop**
A check box for each variable is provided. When a check box is selected by the left mouse button, a check mark can be toggled on or off. The function of the **Drop** column is determined by the option selected in the **Variables** drop-down list. When opened, this list contains three choices:

* **Drop Checked**
  "Checked" variables will be **excluded** from the output database.
**Keep Checked**  Checked variables will be *included* in the output database. Changes the column heading to *Keep*.

**Specify Order**  Choosing this option changes the column heading to *Order* and removes the checkboxes. By clicking the mouse in the boxes in order can specify the order of the variables written to the output database. As you click, an order number will appear. The number automatically increments. If you click on an already numbered box, the number will be removed and all of the higher numbered boxes will be decremented. Only numbered variables will be written to the output database. In other words, order implies keep.

**Type**  This column displays the variable type, either character or numeric. Numeric type variables can be converted to an alternate numeric format. For a list of available formats select the drop-down list associated with each numeric variable type. The standard length of variable types will vary with the output format selected. A description of variable types is provided in the *General Information* section.

**Length**  The length of the variable is displayed in this column.

**Decimal**  The number of decimal places in a numeric variable is shown in this column.

**Label**  For selected database and statistical packages, such as SAS, Prodas, SPSS, and others, an additional *label* field is provided. Labels allow you to associate a character string with the variable name to provide a clearer explanation of the variables meaning. Note: many packages do not support labels. Please see the supported package’s on-line help page to see if your target output package uses labels.

**Format**  This column displays the variables format types. Many packages let you store a display format with the variables. A drop-down list provides access to the following types:
Fixed Formats the variable values with a set number of decimal places.

Comma  number with embedded commas

Aster  Asterisk fill (usually a dollar amount)

E  Scientific notation.

**FormLen**  Sets the entire length of the display format field.

**FormDec**  Sets the number of decimal places for display format field.

**List by Position**
The List by Position check box allows you to change the display sequence of the variables from one sorted in alphabetic order to one sorted by the position of the variables within the database. You can toggle the check box between sort by alpha and sort by position.

**Keep and Drop By Range**
Drop and Keep check boxes can be set in a range instead of one at a time. Check the first box and then with the shift key down, click in the last box in the range. If you now click on one of the highlighted cells, all of the check boxes within the range will be set. Cells can have the checks turned off in the same manner.

**Shrink Sizes**
Scans the entire database to determine if each variable can actually be made smaller in size. For character variables, the length will be set to the longest non-blank line. For numeric variables, the program will determine if it can fit into one of three size integers. The dialog box will be updated to show these values. (Note: this useful for Stata.)
One of the great strengths of **DBMS/COPY Interactive for Windows** is the ability to perform complex translations which change the information passed from the input database to the output database. **The Record Filter & Equations** option from the Power Panel displays the Equation Builder:

The three list boxes, **Variables**, **Functions** and **Goodies**, provide access to the tools of the Equation Builder. Below the list boxes is a large edit area where statements are entered. The **File** area, to the right, enables equations to be retrieved or saved. The **Paste** button allows information selected from any of the three boxes to be pasted into the edit area.

In the **Variables**, **Functions** and **Goodies** categories, when an item is highlighted, information about the item appears in a description field directly below the categories. In the example below, the field variable AMT has been highlighted. As a result, the description field identifies the variable type:
Variables List

The **Variables** list displays variables from the input database and in addition four system variables. The four system variables are:

- `_dateonly_` - A date variable containing today’s date
- `_n_` - A numeric variable containing the current input database record number
- `_pi_` - A numeric variable containing the value of pi.
- `_today_` - A date variable containing today’s date and time.

Goodies List

The **Goodies** list contains a selection of operators and general functions. The description field displays a brief definition of the highlighted item. Detailed descriptions of the available functions and expressions are online by hitting the F1 key. Additional help can also be found within the manual section, *Compute Module Reference.*
Functions

When first displayed, the **Functions** category provides a list of function groups.

The first option listed is **All Functions**. When this option is selected, the function group list is replaced with a complete list of available functions. Selecting any other group function name will display functions specific to the type selected. For instance, if **Character Functions** is selected the Function list displays only character type functions. Note that at the top of the character group list is the option, **Group List**. When a specific group list is displayed, this top-most option, will return control to the initial function groups list. As with the other categories, when a function item is highlighted, a brief definition appears in the description field. Hitting the F1 key when a function is highlighted brings up the on-line help system describing at that function.

General File Handling

To the right of the list boxes, is a collection of action buttons.

The **Paste** action button allows selections from the list boxes to be copied to the instruction edit area, that occupies the lower half of the window. The underlined P in the Paste button indicates that this button can be
activated by pressing Alt+P. (Pasting can also be accomplished by double clicking on the highlighted item).

Selecting the Ok button causes the statements entered in the edit area to be evaluated by the DBMS/COPY and if error-free, control is returned to the Power Panel.

Selecting Cancel exits the Equation Builder without processing the text changes. Control is returned to the Power Panel.

The remaining two action buttons allow previously saved instructions to be retrieved and edited, or a new set of instructions to be saved to disk. When the Open button is activated, the Equation Program File dialog box is displayed. The default file extension for equation program files is prg. The List Files of Type drop-down list box allows alternate file types to be displayed and selected. File selection can be made from the File Name list provided. The Directories list provides the means to search alternate directories for an existing program file. When activated, the Read Only check box marks a program file as read-only. This prevents changes from being made to the file when it is retrieved by DBMS/COPY. A file may be retrieved by double clicking the displayed file name, or by simply highlighting the item and selecting Ok.

In the example above the equation text contains the line:

\[
\text{years_since_review} = \text{year(\_dateonly\_) - year(\^review date\)};
\]

These instructions tell DBMS/COPY to generate a new field, called years_since_review. The new field will be included in the output file along with the original fields. The Equation Editor is closed and instructions kept by selecting the Ok button. Control again returns to the Power Panel.

The results of any calculation entered in the equation builder can be viewed immediately by choosing the View Data option from the Power Panel. This is possible even though we haven't yet generated an output database. We can also examine and modify the variable attributes of our new field by selecting the Variable Information option from the Power Panel.
Selecting Records

The select and delete statements are used to limit the records that are passed to the output database. The select statement defines a filter that each record must satisfy to get passed to the output database. The delete statement defines a filter that each record must satisfy to get excluded from the output database. The keyword select or delete is followed by an expression that defines the filter. For example, to get an output database with all of the employees containing the name of Henry,

\[
\text{select } \text{search(employee_name,'Henry')} \neq 0;
\]

The search function looks in the employee_name variable for the string Henry. If found, the search function returns the starting column. If it is not found, a zero is returned. Here is the same example but expressed as a delete statement,

\[
\text{delete } \text{search(employee_name,'Henry')} = 0;
\]

If you are not sure if the name is all uppercase, all lowercase or capitalized, it can be converted to uppercase first with the following:

\[
\text{select } \text{search(upper(employee_name),'HENRY')} \neq 0;
\]

Expressions can include complex and or operators. For example, the following finds all the Henrys and Joes hired between April 15, 1990 and Feb 3, 1993;

\[
\text{select } \left( \text{search(upper(employee_name),'HENRY')} \neq 0 \text{ or } \text{search(upper(employee_name),'JOE')} \neq 0 \right) \text{ and } \text{hiredate} \geq '4/15/1990'm \text{ and } \text{hiredate} \leq '2/3/1993'm;
\]

Notice the use of parentheses to control the evaluation order. Date constants are entered in quotes with a trailing m.
Creating New Variables

The assignment statement is used to create new variables. The structure of an assignment statement is

\[ \text{variable} = \text{expression}; \]

The `variable` is the name to which you want to assign the value of the `expression`. The `expression` is what DBMS/COPY will compute. For example, to retrieve the year the employee was hired:

\[ \text{hire_year} = \text{year(hiredate)}; \]

In addition to the assignment statement, DBMS/COPY supports the `if-then-else` statement. With if statements you can control which assignment statements are executed. For example, you can translate the values of a coded variable and create a new variable containing human understandable values.

\[
\begin{align*}
\text{if} \ \text{code} &= 1 \ \text{then} \\
\text{readable} &= \text{'Not at home'};
\end{align*}
\]

\[
\begin{align*}
\text{else} \\
\text{if} \ \text{code} &= 2 \ \text{then} \\
\text{readable} &= \text{'Working'};
\end{align*}
\]

\[
\begin{align*}
\text{else} \\
\text{readable} &= \text{' '};
\end{align*}
\]

If statements can control multiple statements with the `do; - end;` construct. For example, we can use the month, day and year functions to generate new variables for all employees hired after 1970.

\[
\begin{align*}
\text{if} \ \text{hiredate} &\geq '1/1/1971'm \ \text{then do;} \\
\text{hire_mon} &= \text{month(hiredate)}; \\
\text{hire_day} &= \text{day(hiredate)}; \\
\text{hire_year} &= \text{year(hiredate)}; \\
\text{end;}
\end{align*}
\]

\[
\begin{align*}
\text{else do;} \\
\text{hire_mon} &= .; \\
\text{hire_day} &= .; \\
\text{hire_year} &= .;
\end{align*}
\]

The above example introduces the DBMS/COPY numeric missing value. The missing value is represented by a single dot (.)

Of course, assignment, if, select and delete statements can all be combined. For example to breakout the month, day and year for the Henrys and Joes:
select search(upper(employee_name),'HENRY') <> 0 or
    search(upper(employee_name),'JOE') <> 0;

hire_mon = month(hiredate);
hire_day = day(hiredate);
hire_year = year(hiredate);

DBMS/COPY can process any number of assignment statements, if
statements, select and delete statements. For a full description of these
statements please see the General Statements documentation in the
Compute Module Reference section. Expressions and functions are also
documented in that section. Numerous other statements are used by the
batch processing part of DBMS/COPY Plus.

**Errors**

When you click the Ok button, DBMS/COPY will check the syntax of
your statements. If there is an error, an error message is displayed and
the equation builder remains active.

The string, *Out_of_Text*, in the error message is telling you that
DBMS/COPY ran out of text while trying to check the syntax. The most
common reason for this error is that a semicolon was left off the last
statement.
DBMS/COPY Interactive - Sort

Selecting the **Sort** button on the Power Panel causes DBMS/COPY to display a dialog box where the sort variables can be specified.

The left hand list shows you a list of the variables on the input database.

Two of the buttons in the middle let you add variables to the Sort By list. Variables can be sorted ascending (smallest to biggest) or descending (biggest to smallest). You can select more than one variable at once. The variables will be inserted before the highlighted variable in the right hand list. Descending sort order variables will have the word (**descending**) appended to the name.

The **Remove** button takes the highlighted variables off of the sort by list.

The database is only sorted after the output package is selected and the transfer request is verified. This dialog box only specifies the sort order.
DBMS/COPY Interactive – Data Information

Ever get a database you know nothing about? Does anyone ever send you a dataset to analyze and the first question you have is, “What’s in the file?” Curious about the contents of those old databases on your computer? Data Information will give you a quick sense of the data. Data Information will display in a spreadsheet format general statistics, minimums, maximums, percentiles and discrete values for each variable.

After you click the Data Information button on the Power Panel the following dialog box will come up.

From here you can control the following parameters:

**Variable Limiting**  
This initial variable selection dialog box is used to specify the variables you want the results to be limited to. If you want the analysis done on all variables, then don’t pick any. The default is all variables.

**Max # of Values**  
Limits the maximum number of discrete values which are displayed to this number. If for example, you have a million record database and each record has a unique identify it is not
informative to see a spreadsheet with a million rows. If the variable has more this number of values, the last line of the spreadsheet will tell you.

**Percentile Step**

Set how often you want percentiles. If the value is 25 then you get 25, 50, 75 and 100.

After you click Ok on this dialog box, DBMS/COPY will scan your database and show you a spreadsheet with the results.

## Data Information Results

The Data Information results are displayed in a scrollable spreadsheet.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CITY</th>
<th>COMMENT</th>
<th>COUNT</th>
<th>BIRTHDATE</th>
<th>DEPT</th>
<th>CODE</th>
<th>NAME</th>
<th>GENDER</th>
<th>RACECODE</th>
<th>PENDATE</th>
<th>RACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>114 VALLEYSIDE</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123 BAY STREET</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124 MAIN STREET</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130 PARK AVENUE</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each variable for which calculations were generated will have two columns on in spreadsheet.

The first column is the number of records containing the value. The count column always precedes the results column.

The results column shows:
Records number of records read
Count number of records with data
Missings number of records with missing data
Mean average value
Sum total of all non-missing values
Range Maximum minus minimum
Std Dev Standard Deviation
Std Err Standard Error
Min-# 5 lowest values (not 5 lowest records)
Max-# 5 highest values (not 5 highest records)
Mode Value with most number of hits (if tie the lowest value)
Percentile Requested percentiles
Value-# Unique values sorted lowest to highest. Preceded by count
More Values? If more values exist than the limit this will have a yes.

Write To Database
Save the results to an output database. The output database will have the following two variables added automatically:

_stat_ text string with statistics name.
_index_ increment for the value# and percentile#

For each input variable you will get two output variables. The counts will be written variables with a c_ prefix.
Interactive Options

DBMS/COPY maintains a few global options. After selecting the Options menu item under the Interactive heading, the following dialog box will appear.

Output Dates Always use 4 digits
Some software packages do not maintain a "date" variable type. For those packages, DBMS/COPY will write the date as a character string.

If this box is not checked, the dates will have 2 digit years if the date is in the current century and will have 4 digits if not.

If this box is checked, the dates will always have 4 digits.

Input Database 2 digit year start
A few input packages only store the year in two digits. With the new century coming, the question becomes what does a year 02 and a year 98 mean? This option lets you define the base for the 100 year range for what a two year means.

For example, if the base is 1950. The following examples will make this clear.

Year 02 -> 2002
Year 98  ->  1998
Year 50  ->  1950
Year 49  ->  2049

Shrink Variable Size Automatically
If checked, the shrink variable size feature (as described on the Power Panel’s Variable Information dialog box) will be run automatically when an input database is selected.

Empty Log  Sets the log file size to zero bytes.
DBMS/COPY Interactive - Writing Multiple Member Files

Some software packages (DataEase, BMDP, Rbase and others) store multiple sets of data within one disk file.

The following dialog box is used to enter the name of the output member name. The name can be selected from the list or entered by hand in the text box. If the name matches one already on the disk file, you will be asked if you want to overwrite the member.
DBMS/COPY Interactive - Writing ASCII files

DBMS/COPY can write data to free format and fixed format ASCII files. Free format files have each field separated from the next field by a single delimiter. Fixed format files have each field occupying the same set of columns on each record.

The ASCII Output Format Options dialog box will appear after the filename was specified on the Output Database dialog box.

The first task is to identify if the output file is to be fixed or free format.

**File is Fixed Format**

The check box toggles between a list of options for fixed format files and a list of options for free format files. You should first set the state of the check box before moving on to the other options. If a fixed format file is desired, the check mark should be displayed. The dialog box shown above, displays the options for free format files.
Fixed Format File Options

When the File is Fixed Format box is checked, the dialog box will appear as follows:

![ASCII Output Format Options](image)

Shared Fixed & Free Format Options

**Date Value Format**

The drop-down list displays commonly used output date formats. When date formatted variables are written to the ASCII file, the format selected here will determine how **DBMS/COPY** writes those values. The month, day and year date components can be separated by special characters, such as the forward slash or hyphen. Embedded spaces are not permitted.

**Numeric Missing Value**

Identifies the string that DBMS/COPY will write for a numeric missing value. For example, you may want -999 as a missing value.

Free Format Options

**Field Separator**

The field separator names the character to be placed between fields on the output ASCII file. Common field separators are comma, space and tab. To use a space or
 comma, simply enter the key itself. To use a tab, simply enter is its ASCII numeric value, which is a 9. The default provided is the comma.

**Surround Character**

In certain cases, the field separator character may be part of valid data within fields. If the space character is defined as the field separator, then a name consisting of a first name and a last name would be divided into two fields. To avoid this problem, the **Surround Character** option tells DBMS/COPY what character to use to surround fields containing the separator character. Frequently a double quote is used.

**Variable Names on Row #1**

If this check box is selected it tells DBMS/COPY that the variable names should be written to the first line of the ASCII file.

**Every Field Must Be Surrounded**

If this check box is selected it tells DBMS/COPY that every field should be surrounded by the surround character even if it does not contain the field separator character.

After you are satisfied with the choices, click the Ok button.
DBMS/COPY Interactive - 
Writing Microsoft Access Tables

DBMS/COPY can write the output data to a Microsoft Access table. The table can be written to an existing Access database or DBMS/COPY can create a new database. The data can be written to a new table, replace an existing table or appended to an existing table.

After selecting the Access database in the Output Dialog Box. The following dialog box is displayed:

You are shown a list of existing tables in the Access database. If you check the **Append to Existing Table** box, the table must already exist. If you select an already existing table, you will be asked if it is alright to delete that table before writing over it.
DBMS/COPY Interactive - Writing ODBC Tables

DBMS/COPY can write the output data to ODBC supported packages. All of the data is written to a single table. The data can be appended to an existing table, written to a new table or replace an existing table.

The ODBC output system is selected by hitting the ODBC SQL button on the Output Database dialog box. After logging in to the ODBC system. The ODBC Output table dialog box is shown below:

You are shown a list of existing tables in the database. If you check the Append to Existing Table box, the table must already exist. If you select an already existing table, you will be asked if it is alright to delete that table before writing over it.
DBMS/COPY Interactive - Transfer Verification

The Transfer Verification dialog box is shown after selecting the input database, using the Power Panel and specifying the output database.

The DBMS/COPY Plus batch file equivalent of the transfer you create is shown in the large scrollable text window. Below that window are three buttons:

Do-It! Executes transfer. During the transfer a processing progress window will be displayed.

Cancel This button will backup to the output database dialog box. You can use the Cancel button to backup all the way to the input database.

Save Program Transfers that will be needed again can be saved to a disk file. The program must be saved before clicking the Do-It! button. The disk file can then be run and/or modified in the DBMS/COPY Plus batch processing facility. If you do not save, the specifics of this transfer will have to be reentered next time.
The Command Syntax

Except for ODBC files, simple input/output transfers can be specified on the command line. Thus, transfers can be executed from the Start/Run menu item. The names of the input and output files follow the application name. Also, the transfer can be made into an icon or executed from inside a 3rd party Windows shell. The syntax follows this model:

```
dbmswin7 source.ext target.ext
```

`dbmswin7` This is the executable name for DBMS/COPY for Windows. In order for Windows to recognize this command, the file `dbmswin7.exe` must be fully qualified with drive and directory. For example, `c:\dbmscopy\dbmswin7`. For these examples, we will use the `dbmswin7.exe`. Note: the name has changed since DBMS/COPY Version 6, if you have routines which depend on the name being `dbmswi32.exe`, just make a copy of the `dbmswin7.exe` to `dbmswi32.exe`.

`source.ext` The name of the existing file you wish to translate to another format. The `ext` (pseudo-extension) must be specified so that DBMS/COPY can identify the source system. The extension may or may not be the actual file extension for the input file. The source should also contain the path.

`target.ext` The name of the new file to which you will be translating the source data. The pseudo-extension must be specified. The target should also contain the path.

File Extensions

DBMS/COPY uses the file extension to identify the software product and file structure with which it will be working. Since many different products may use the same disk file extension, DBMS/COPY uses a pseudo extension as part of the command line. Therefore, the file extension you will use with the `dbmswin7` command might be different from the actual file extension.
The on-line help for the products name the file extension to use for each product. For more information on pseudo extensions see How DBMS/COPY Works.

The file extension specified on the command line is critical. If the wrong extension is entered, DBMS/COPY will not be able to complete the data transfer as desired.

**Spreadsheets**

Spreadsheets consist of rows and columns. The rows may be thought of as the records in a database system, and the columns as variables. If thought of in this fashion, a spreadsheet may be used as a database.

The process of reading a spreadsheet is different because part of the spreadsheet contains data, part contains the variable names and part may contain other miscellaneous information. The variable names may be next to the data, or may not even be included. With the command line interface, the user must either specify the locations of the data and the variable names or allow DBMS/COPY to use its default assumptions.

**Default Settings**

If there are no user defined settings, DBMS/COPY assumes the spreadsheet is in the default format with variable names in row 1 and records beginning in row 2.

When DBMS/COPY reads the first row of an input spreadsheet it expects to find character strings in each column, and treats these character strings as variable names. All characters, including embedded blanks, are permitted in the variable names. If DBMS/COPY encounters columns without a character string in the first row, these columns are not read.

**User Defined Settings**

In most cases, the format of a spreadsheet will not exactly match the requirements of the default settings. Variable names may not be present, may use more than one row, and there may be blank rows or other dividers between the variable names and the first row of data.

DBMS/COPY has the flexibility to handle all these variations. If you are not familiar with the spreadsheet you need to translate, it is
recommended you use the menu system since it allows you to review the spreadsheet contents to identify variable names and data rows.

If you are familiar with the spreadsheet, and you prefer to use the command line, all you need to do is identify the cells that contain the data and the rows that contain the variable names.

To tell DBMS/COPY where the information is, you specify on the command line the data range and the variable name rows in brackets. The information is put in brackets and placed immediately in front of the spreadsheet file name. The brackets contain, in this order,

- the top row of the data range
- the bottom row of the data range
- the left column of the data range
- the right column of the data range
- the top row of variable names
- the bottom row of the variable names

For example, the ar.xls sample spreadsheet has several lines of headings. The data is contained in a range beginning with Row 9, Column A and continuing down to row 18 and over to Column M. The variable names begin in Row 7 and continue in Row 8.

For the example above, the command line would be

```
dbmswin7 [9,18,A,M,7,8]ar.xls outfile.ext
```

where `dbmswin7` should be fully qualified with the drive and directory and `outfile.ext` is the necessary description of the destination file.

If the variable name rows are given as 0, then DBMS/COPY will assume there are no variable names and assign A, B, etc. as default variable names.

For spreadsheets with no variable names and data starting in cell 1A., put [1,0,0,0,0,0] before the file name. DBMS/COPY will read the entire contiguous data range and use A, B, C, etc. as the variable names.

**Log File**

Sometimes DBMS/COPY must change the names of the variables in the destination file in order to comply with the requirements of the destination software package. (For more information about these name
changes, see *How DBMS/COPY Works*.) Whenever this occurs, **DBMS/COPY** records the changes in a file called `dbmscopy.log`. After the file transfer is complete, you can review the contents of this file using the DBMS/COPY log file viewer. This file is appended to with every transfer.

**Sample Files**

Several sample files have been provided so that you can follow along with the examples below and elsewhere in the manual. Make sure that you add the appropriate directory path to the examples that follow.

The sample files are: *mail.db*, a Paradox file; *mail.dat*, an ASCII file; *fit.sys*, a SPSS/PC+ file; *ar.xls*, an Excel Version 4 file; *employee.rxd*, a Reflex file; and *ar.wk1*, a Lotus file.

It is not necessary to own any of the systems used in the samples in order to use these examples.

**Examples**

We can move *mail.db*, the sample mailing list database written using the Paradox DBMS system, to the Reflex format using the following command sequence:

```
  dbmswin7 mail.db mail.rxd
```

The sequence begins with the command, `dbmswin`. Next the source database is specified, along with its extension (*db*), so that **DBMS/COPY** can determine the most appropriate translation routines. Finally, the target database is named along with its extension (*rxd*), so that the data is translated correctly.

The Reflex database can then be moved to a Lotus spreadsheet:

```
  dbmswin7 mail.rxd c:\lotus\newmail.wk1
```

The patient fitness database, a SPSS/PC+ file, can be moved easily into a dBaseIII format using the following command:

```
  dbmswin7 fit.spss k:\dbasefiles\temp\fit.dbf
```
DBMS/COPY Plus
Creating Batch Programs

DBMS/COPY Plus provides a method of storing transfers for execution again at another point in time. These "batch" files can contain any number of transfers and can include sophisticated record filtering, new variable computation, variable selection and variable renaming. For example, if you need to translate 30 files every Friday, create a batch file in DBMS/COPY Plus and execute it with one command.

Now that DBMS/COPY Version 7 includes a free license to DBMS/Analyst, the batch world is greatly enhanced with addition of a sophisticated multiple database programming language, macro language, basic statistics and tabulation. If your task would benefit from the power of DBMS/Analyst, please see the section on DBMS/Analyst in this manual and the complete online DBMS/Analyst manual for more information. This section of the manual will discuss only those routines which are part of the DBMS/COPY batch system.

The Easy Way To Build A Batch File
Instead of learning the DBMS/COPY syntax, you could use the DBMS/COPY Interactive interface to define the transfer and save it to disk. The Transfer Verification window shows the DBMS/COPY Plus batch file equivalent to the interactive transfer. If at that time you click on the Save Program button, a save file dialog box will give you an opportunity to save the text to a disk file.

Syntax Overview
The required syntax for DBMS/COPY commands is discussed in detail in the Compute Module Reference Section later in this manual. However, a brief overview is provided here.

DBMS/COPY Plus commands are called statements, and a group of DBMS/COPY Plus statements is called a paragraph. The standard form of a DBMS/COPY Plus paragraph is:
On the paragraph above, the `module_name` is replaced by the name of a DBMS/COPY module. The DBMS/COPY Plus system consists of four modules: `compute`, `contents`, `datainfo` and `sort`. The Compute Module is the main module for transferring data between software systems. The Contents Module displays a list of the variable names on a specified database. A general paragraph using the Compute Module is given below:

```plaintext
compute;

statements

run;
```

The run; statement ends the paragraph and tells DBMS/COPY Plus you are finished giving commands. The statements in between the compute and the run statements include record selections, new variable computation, and variable selection. Each statement ends with a semicolon.

DBMS/COPY Plus can execute a paragraph from the menus (Run under File) or on the command line.

Syntax Example

In the batch paragraph below, a dBase file is translated to Lotus format. Only records which match the specific requirements set forth in the `select` statement are included in the output. In this example the invoice file, SALEDBF.DBF, has been retrieved.

In the Compute Module, the first command after the module name is the Database Specification Statement which contains the names of the input database and output database.
The in= and out= statements indicate the path and name of the input and the output database files, respectively. The select statement, which includes the logical and operator, defines how each record will be evaluated. In this case, only those records which contain "hlf" in the SALESMAN field, "BOOKS" in the PRODUCT field, and a DATE value within the month of June, will be included in the output file. Finally, a new field is created in the output database. Each qualifying record will include the field COST_PER_UNIT, contains an average per unit cost for each invoice.

Included in the expressions above are the functions upper() and round(). Functions are built-in instructions designed to perform a specific task. The upper() function converts character variable data enclosed within its parenthesis to upper case. The round() function returns the numeric value contained in its first parameter, amt/units, in its rounded form, based upon the number of decimal places indicated by the second parameter, .01.

A comprehensive description of statements, expressions and functions used in DBMS/COPY may be found in Compute Model Reference section.
The Batch Editor is an integrated editor for the creation of DBMS/COPY Plus batch files. The editor’s built-in Assistants ease the task of creating the files. Assistants exist to automatically build the input database name and the output database name. Once an input database is named, another assistant makes it easy to paste variable names into the program text. Help is always available with the F1 key or the Help Menu. There is general help on using the editor and direct tie-ins to the statement and function library help systems.

File Menu
The process of creating or editing a batch program begins at the File menu:

```
<table>
<thead>
<tr>
<th>File</th>
<th>Interactives</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Batch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Batch...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Batch...</td>
<td>F9</td>
<td></td>
</tr>
<tr>
<td>Log File</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit</td>
<td>Alt+F4</td>
<td></td>
</tr>
</tbody>
</table>
```

Two menu items, New Batch and Open Batch, access the batch editor. The third menu item, Run Batch, retrieves an existing batch program and immediately executes it, without opening the editor. When the editor is opened, options within the File menu are expanded to the following:

```
<table>
<thead>
<tr>
<th>New Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Batch...</td>
</tr>
<tr>
<td>Save</td>
</tr>
<tr>
<td>Save As...</td>
</tr>
<tr>
<td>Log File</td>
</tr>
<tr>
<td>Print...</td>
</tr>
<tr>
<td>Run</td>
</tr>
<tr>
<td>Save &amp; Run</td>
</tr>
<tr>
<td>Close</td>
</tr>
</tbody>
</table>
```
The **New Batch** and **Open Batch** menu items create another batch editor window. (There can be any number of open windows.) The **Save** and **Save As** saves the contents of the batch editor window to a text file. **Save & Run** saved the contents of the batch editor window and then run the batch file. The **Run** menu item runs the batch without first saving the window. The **Log File** menu item displays the log file in the log file window. The log file is updated with the results of executions of the batch file.

After opening an edit window, the menu bar will now include the following items:

- **Edit** manage the edit window
- **Search** find and replace text
- **Assistants** automatic generation of major text elements
- **Interactives** run the Interactive interface
- **Window** window management
- **Help** access to the help system
Edit Menu

The Edit menu provides access to window text management tools. The editor is a simple text processor. The Edit menu provides options for cutting and pasting, changing the font size or style, and an Undo option for restoring the most recently changed text. When activated, the Select All option will highlight the contents of the active window.

<table>
<thead>
<tr>
<th>Edit</th>
<th>Search</th>
<th>Assist...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Ctrl+Z</td>
<td></td>
</tr>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
<td></td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
<td></td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>Del</td>
<td></td>
</tr>
<tr>
<td>Select All</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Font...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many options within a DBMS/COPY menu have associated hot keys. Using an option’s hot key will activate the menu option without having to open the menu itself. For instance, pressing Ctrl+V will paste text that had been previously cut or copied to the Windows clipboard.

All batch program files generated by DBMS/COPY are in ASCII text format. In certain instances a user may prefer to use their own text editor. Any "cut and paste" operation performed in DBMS/COPY will place text on the Windows clipboard. This material can be retrieved by a separate Windows text editor by switching to the text editor and "pasting" it from the clipboard.

Search Menu

The Search menu contains options for finding and replacing text. When the Find... menu item is activated, a dialog box allows the search text to be entered. F3 is a hot key that enables the user to quickly repeat a text search. Replace... presents a dialog box that accepts both the text to search for and the text to replace it. The user has the option of searching for and replacing each occurrence of the text one at a time with verification, or simply doing a global search and replace on all matching text within the program file.
Assistants

The Assistants menu automates the writing of many of the common statements needed when building a batch program. The Input Database... option presents the Input Database dialog box, identical to the dialog box presented when the Copy Database... option is selected from the Interactives menu. However, when this option is selected from within the batch editor, the batch instructions for the selected database are inserted into the editor. Similarly, the Output Database... option display the Output Database dialog box, and inserts the appropriate instructions into the edit window. Once an input database has been selected, the Variable List... option presents a list of the variables contained within the database.

When activated, the Compute Module... option within the Assistants menu initiates a process that generates the basics of a batch program. When selected, this option displays the Input Database dialog box and then the Output Database dialog box, in sequence. When this process is completed, the information inserted into edit window appears similar to:

```
compute;
in=c:\dbmscopy\files\sales.db
     out=c:\dbmscopy\files\sales.clarion;
run;
```

In the above example, a Paradox file called SALES.DB was named as the input database. The output database has the same file name, but in Clarion format. The program also inserted the commands compute and run, to complete this "bare bones" batch program. It is now easy to insert additional statements to tailor the translation.
Variable List Assistant

When activated, the Variable List... option within the Assistants menu displays a list of input databases and a list of the variables for each database. Only databases known to the system by the Input Database... menu item, will be accessible by the Variable List dialog box.

By highlighting a variable name within the Variable List dialog box and clicking the Paste button, variable names are immediately copied into the editor window. Multiple variable names can be selected and pasted. For example, the keep statement names variables that should be written to the output database. You could type the keyword keep and then bring up the Variable List dialog box to select the variables. Don't forget to add the trailing semicolon.

Window Menu

The Window menu provides services for moving and sizing multiple edit windows. In cases where it is necessary to compare or duplicate instructions between batch programs, the options contained in this menu will make this task easier.
DBMS/COPY Plus – Using The Command Line

Just as DBMS/COPY Interactive can be executed with a one line command, so can DBMS/COPY Plus. The DBMS/COPY Plus command line is:

```
dbmswin7 PLUS file.prg
```

`dbmswin7` is the fully specified executable name for DBMS/COPY.

`PLUS` says this is a Plus batch file execution and not a simple input/output command line transfer. Must be uppercase and must appear as the first parameter. Remember the simple input/output command line looks like:

```
dbmswin7 source target
```

`file.prg` is the name of a batch file containing the DBMS/COPY Plus commands.

The following would run the bigtrans.prg program file:

```
c:\dbmscopy\dbmswin7 PLUS bigtrans.prg
```

The above command can be executed from the File/Run menu item within the Program Manager or from within any Windows Shell program. The command can also be turned into an icon.

Batch files for DBMS/COPY Plus are text files containing one or more Plus paragraphs. They are different from Windows and DOS batch files because they can only be executed by DBMS/COPY Plus and they do not contain any Windows/DOS commands.

When DBMS/COPY Plus is instructed to process the batch file, it executes the paragraphs, in order. There is no limit to the number of paragraphs that can be included in one batch file. All of the module executions within that batch file are executed.
Batch File Error Checking
If DBMS/COPY Plus encounters an error when processing a batch file, it will write the error message in the log file and stop.
A continual record of *DBMS/COPY* activity is posted to a log file maintained by the program. The file name for the log is `dbmscopy.log`. It can be found within the main DBMS/COPY directory. This record is viewed by selecting the **Log File** option from the **File** menu:

```
+++ 49 Records Of 7 Variables And 99 Bytes Written To $:\DBMSCPY\TABULATS\FILES\employee.ascii
---

*** Input From Interactive Copy  Tue Sep 16 14:42:32 1997
compute;
  input:\text{Tabulate}\text{Files}:employee.ascii;
  output:\text{Tabulate}\text{Files}:employee.ascii;
  years_since_review = year(\text{dateonly}) - year("\text{review date}");
  order depart rating jobtitle salary years_since_review years_time since_review_date;
  label depart=\text{Department};
  label jobtitle=\text{Job Title};
  xtype=years_short;
+++
The Following Notes Apply To Output SAS/Windows Database -> $:\DBMSCPY\TABULATS\FILES\employee.ascii
  Variable Name Change: years_since_review -> YEARS_S
  Variable Name Change: review_date -> REVIEW_D
+++ 49 Records Of 7 Variables And 99 Bytes Written To $:\DBMSCPY\TABULATS\FILES\employee.ascii
---

*** Input From Interactive Copy  Tue Sep 16 16:14:55 1997
```

A log entry is created whenever a **DBMS/COPY** attempts a file translation. The example above depicts a single log entry. The user executed a simple translation using the **Copy Database...** command from the Interactives menu.

The first text line of the log indicates the source of the commands. In the above example the log file indicates an Interactive Copy was executed. Had the same procedure been generated by a batch program within the editor, the first line of the log file would appear as:

```
*** Input From Editor File At Line=1  Mon Sep 01 17:30:33 1997
```

After the first line, the remaining statements for that module execution are listed.

Prior to running the program, **DBMS/COPY** checks the batch instructions to insure that all statements have the correct syntax. If error is encountered, an error message would be written to the log file. For example:
The line beginning with the error number explains the error. In this case, the semicolon has been inadvertently left off the run statement.

The log file maintains a complete listing of interactive and batch executions.
DBMS/COPY Plus

Plus Modules - Overview
The equation builder within **DBMS/COPY Interactive** can use the *If, select, delete* and *stop* general statements, expression processing and the large function library.

**DBMS/COPY Plus** uses all features in this section.

**DBMS/COPY Plus** consists of four different modules, Compute, Contents, Datainfo and Sort.

Compute The Compute Module is used to transform a database from one software package to another software package. While transferring, you can select records, select variables and compute new variables.

Contents The Contents Module lists the names of the variables on a database.

Datainfo The Datainfo module generates an output database showing the general statistics, percentiles and values for each variable.

Sort The Sort module creates a sorted database.

**DBMS/Analyst**
DBMS/COPY Plus has been enhanced with the inclusion of a free license to **DBMS/Analyst**. DBMS/Analyst enhances the power of DBMS/COPY with a powerful multiple database manipulation and programming system. DBMS/Analyst helps clean and scrub data. DBMS/Analyst is briefly described later in this manual. The complete DBMS/Analyst manual is included on-line.
Syntax

Command Form

A DBMS/COPY Plus batch file consists of one or more module executions. A module execution is also referred to as a paragraph. Each paragraph begins with the module name, contains one or more statements, and ends with a Run Statement.

The syntax for a module execution is

```
    modulename;
    statements;
    run;
```

All statements, including the module name statement, must be followed by a semicolon. Since DBMS/COPY Plus looks for a semicolon to indicate the end of a statement, the statements can be entered in free format.

The last command in a DBMS/COPY Plus module execution must be the Run Statement. It tells DBMS/COPY Plus that you are finished entering commands into the module and that DBMS/COPY Plus should now execute the module.

DBMS/COPY Plus batch files can have more than one module execution.

There are no reserved words in DBMS/COPY Plus, but certain words are considered keywords. These are words that have a specific meaning in the proper context. For example, run is a keyword. When run is the only word in a statement, DBMS/COPY Plus knows that it has received all the instructions for the current module and it is to execute the statements. In different contexts, run might be the name of a variable or the character string stored in a variable.

Blanks are only required to separate variable names from keywords where, if no blank was present, the two symbols would be confused. For example,

```
    if amount = 3
```

cannot be written as

```
    ifamount = 3
```
because this would assign the value 3 to a variable named *ifamount*. It could be written as

```plaintext
if amount=3
```

because there is no potential for confusion. (If a variable name contains an equal sign (=) the name must be surrounded by caret marks (^); so *amount=3* cannot be a variable name.)

Any place that one blank is allowed, any number of blanks can be used. For example,

```plaintext
if amount=3
```

Also, any place that one blank is allowed, any number of new lines are allowed. The *DBMS/COPY Plus* software system converts a new line into a single blank when executing the paragraph. For example,

```plaintext
if
  amount = 3
```

Embedded comments are also allowed wherever a blank is allowed.

*DBMS/COPY Plus* does not differentiate between upper and lower case letters in paragraph statements. Run is interpreted the same way as RUN and run are.
Embedded Comments

The DBMS/COPY Plus software system can have comments embedded within the paragraph at any place where a blank can be entered.

A comment starts with a /* and ends with a */.

Examples,

```
in=abc /* this is a comment */ ;
in=abc;   /* This is a comment */
```

It does not matter whether the comment comes before or after the semicolon. Comments cannot be nested.

Variable Name Shorthand

Many statements in the DBMS/COPY Plus system allow a shorthand notation for the variable names. Wherever the short hand notation for variables can be used, it will be stated explicitly in the manual chapter covering that statement.

If the variables have names that are in a sequence, such as test1 test2 test3 test4 test5, they can be represented by a shorthand notation, test1-test5. When executing the paragraph, DBMS/COPY Plus expands the notation and treats the statement as if it were test1 test2 test3 test4 test5.

The notation is the first variable name, a dash, and the last variable name. The last part of the variable names must be numbers in sequential order and the characters before the sequential numbers must be identical. For example, abc3-xyz5 is not valid.

It is not necessary for the numbers to begin with 1 and there is no limit to the number of variables that can be included in the shorthand.
DBMS/COPY Plus - Error Messages

The DBMS/COPY Plus software system generates an error message when the input program does not agree with the syntax for the module you are executing. The error messages are written to the logfile.

The error message display includes:

- module name
- input source of the error
- line number
- column number
- error message text
- error message number

The message is a one line description of the error.

The line and column numbers inform you that the error occurs before the line and column (usually the problem ends at the line and column given).

Since DBMS/COPY Plus is a free format system and without reserved words error messages must be carefully interpreted. For example, if the following two correct statements

\[
x = 2; \\
y = x;
\]

were written without the semicolon on the first statement

\[
x = 2 \\
y = x;
\]

the error message generated would be:

```
Error #3: Symbol out of place in assignment statement -> y
```

Since the semicolon was left off the first assignment statement, DBMS/COPY Plus thinks the second statement is just another part of the first Assignment Statement. To DBMS/COPY Plus this looks like

\[
x = 2 \ y = x
\]

The \(y\) has not been defined yet, so it cannot be used on the right side of the equal sign of an Assignment Statement. That would be using an
undefined variable to define another variable. Error messages must be viewed in the context of the actual program lines read by DBMS/COPY Plus.

The Error messages and suggestions are in the on-line help system.
Compute Module

Purpose
The Compute Module allows users to access the DBMS/COPY Plus record filtering, expression processing, function library, and condition computation while transferring records from the input database to the output database.

Syntax
compute;
    in=inbase    out=outbase;
    options shrinkvars;
    statements;
    run;

Syntax Elements
compute; The first statement for all paragraphs must be the module name followed by a semicolon; in this case, compute;.
in=inbase    out=outbase;
    For the Compute Module, the second statement must be the Database Specification Statement and must contain both the in= and out= components. This statement identifies the input and output databases.

    If the in= and out= name the same database, DBMS/COPY Plus will overwrite the data in the input database with the results of the module execution. Otherwise, DBMS/COPY Plus will not make any changes to the input database.

    Within DBMS/COPY Plus, Access and ODBC input specifications uses the inodbc statement and the output specification uses the outodbc statement. For more information please see these two statements in the General Statements chapter. ODBC (Open DataBase Connectivity) is the method for accessing SQL databases.

    For more information on the Database Specification Statement, see the General Statements chapter of the manual.
Options shrinkvars;
   Automatically shrink the size of the variables when creating the output database. Please see the Power Panel’s Variable Information Dialog Box for a more detailed description of the shrinking algorithm.

Statements There can be any number of additional statements following the in= ... out= statement. Allowable statements for the Compute Module are the statements listed under General Statements in the Compute Module Reference section of the manual. Each statement must end in a semicolon.

run; The last statement of all paragraphs must be run; . It tells DBMS/COPY Plus there are no more commands, and DBMS/COPY Plus should now execute the paragraph.

Usage and Examples
The discussion below will provide examples of several different DBMS/COPY Plus statements. The exact syntax of each statement is given in the Compute Module Reference Section of the manual.

At its most basic, the Compute Module reads in one database and translates it to another. That is, it can function exactly like DBMS/COPY. For example

compute;
   in= employee.rxd out= employee.db;
   run;

will translate the employee Reflex file to a Paradox file of the same name. The power of DBMS/COPY Plus lies in its ability to manipulate and customize the output file.

One feature of the Compute Module is to create a new database containing only part of the input database. The new database can be in the same software system as the original database, on in a different one. This feature allows the user to translate only the information that is needed.

DBMS/COPY Plus can take the employee database (Reflex) and create another database (Paradox) with just the males.
compute;
in=employee.rxd out=males.db;
select sex = 'm';
run;

The original employee database is unchanged, but the new employee database (in Paradox) will contain only the records for the men. Select and Delete are general statements used to filter records.

Another example is to keep all the records but transfer only a few of the variables.

compute;
in= employee.rxd out= deptsal.rxd;
keep depart salary jobtitle;
run;

Once again, the original Reflex file is unchanged. The new file is also a Reflex file. It contains as many records as the original, but each record contains only three variables, depart, salary, and jobtitle. The general statements for filtering records are Keep and Drop.

The Compute Module can also create new variables using DBMS/COPY Plus’s sophisticated expression processor and function library. For example

compute;
in= fit.spss out= fit.ncss;
logpulse = log(runpulse);
run;

This DBMS/COPY Plus paragraph will translate the SPSS/PC+ database called fit to an NCSS file with the same name. The NCSS file will contain all the information as the original file as well as a new variable called logpulse. The new variable will contain the natural logarithm of the value in the runpulse variable. New variables are created with an Assignment Statement which uses an equal sign to assign a value to the new variable.
The function library can also be used with any of the other DBMS/COPY Plus statements. For example, the following paragraph will create a new file containing only the records of engineers with ten or more years of employment. All employees with titles that include the word engineer will be identified.

```plaintext
compute;
in= employee.rxd  out= engineer.dbs;
select index(upper(jobtitle),'ENGINEER')
   and years >= 10;
run;
```

In addition to the statements used previously, this example uses the logical and operator as well as two functions, index and upper. The upper function converts the data in the jobtitle variable to all capital letters for the purpose of comparing it to the word ENGINEER. This is so that if some of the entries contain capital letters and some contain lower case letters, they will still be properly identified. The index function will look for the second character string (ENGINEER) in the first (the data in the jobtitle variable). If the string is present, the column number where it begins will be returned. If not, a 0 will be returned. A 0 is the equivalent to the logical false and any other number the logical true. The and operator will consider both the result of the index function and the relational expression (years >= 10). If both are true, the record will be selected and sent to the output file. If either fails, the record will be skipped.

This example is a little more complicated than the others because of the nesting of functions and the logical operator. However it is still fairly straightforward.
Assignment, Delete, and Select Statements can be conditionally performed. That is, one statement is executed if certain conditions are met and another if they are not met. For example,

```
compute;
  in= employee.rxd out= employee.dbf;
  if years >= 10 then
    tenured = 't';
  else
    tenured = 'f';
run;
```

This paragraph translates the Reflex file to dBase. The new file will contain a new variable called `tenured`. The `tenured` variable will be a character variable. The records for employees with 10 or more years of employment will have `t` in the `tenured` variable. The records for employees with fewer than 10 years of employment will have `f` in that variable. If / Then statements are used to conditionally perform other statements.

These examples have all been fairly simple. The tasks that DBMS/COPY Plus can perform can be much more complex, but they will be built on the same principals as these examples. There are several other general statements that can be used in the Compute Module. There can be any number of general statements in any `compute` paragraph.

**Executable Statements**

The Assignment Statement and Drop Statement illustrate the two different types of DBMS/COPY Plus statements that effect the output database and its variables. The types are executable and nonexecutable statements.

An executable statement is one that is executed for each record. New variables may be calculated or the record itself may be included or excluded. The Assignment Statement is an executable statement. In the last example, a new variable was calculated for each record. Other executable statements are If / Then, Delete, and Select.

A nonexecutable statement is one that sets an environment or condition that applies to the entire database. The Drop Statement is a non executable statement. One example established that the `depart` variable would not be written to the output file. Other nonexecutable statements are the Database Specification Statement, Exit, Keep, Label, Modify, Obs, Rename, Retain, Run, Varlist, and Vartype.
Executable statements can be conditionally executed. It is not possible to include nonexecutable statements in If / Then Statements.

Some of the nonexecutable statements can be used as options in the Database Specification Statement rather than as separate statements. This will speed up the processing of the file.
Contents Module

Purpose
The Contents Module prints a list of variables that are on a specified database. The listing is appended to the log file.

Syntax
contents;
in=database;
short;
position;
run;

Syntax Elements
contents; The first statement for all paragraphs must be the module name followed by a semicolon; in this case, contents;.
in=database; Names the database for which a variable list should be printed.
short; Short abbreviated listing. (optional)
position; Sort the listing in the order that the variables appear on the database. The default is alphabetic sort. (optional)
run; The last statement of all paragraphs must be the run; statement. It tells DBMS/COPY Plus there are no more commands, and DBMS/COPY Plus should now execute the paragraph.

Example
The following example shows that the contents module does:

contents;
in=mail.db;
run;

The following listing shows the number of variables and records on the database and a list of the variables.
** * * Database Contents Listing * * *

Database Name : mail.db  
Database Driver : paradox2  
Number of Variables : 7  
Number of Records : 4  

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Char(30)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>Short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>Char(15)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maildate</td>
<td>Date</td>
<td>mdy.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Char(20)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Char(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zip</td>
<td>Char(5)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Datainfo Module

**Purpose**
The Datainfo Module creates a database of summary statistics for the variables on the database. The statistics are written to the output database. Beyond the options and var statements, the module is exactly the same as the compute module. The datainfo module can include record filters and new variable computations.

**Syntax**
```
datainfo;
in=inbase       out=outstatistics;
var variablelist;
options value_count=#   percentile_step=#;
statements;
run;
```

**Syntax Elements**
- `datainfo;` The first statement for all paragraphs must be the module name followed by a semicolon; in this case, `datainfo;`.
- `var variablelist;` The keyword `var` can be followed by any number of variables on which statistics will be generated. If no `var` statement is given, all variables will be used.
- `Options value_count=#   percentile_step=#;` modify the default limits.
  - `Value_count=#` Limits the maximum number of discrete values which are calculated to this number. If for example, you have a million record database and each record has a unique identify it is not informative to generate an output database with a million rows. If the variable has more this number of values, the last line of the spreadsheet will tell you. The default count is 1000.
Percentile_step=#
Set how often you want percentiles. If the value is 25 then you get 25, 50, 75 and 100. The default step is 1

Usage
In the following example, the percentiles will be limited to 25, 50, 75 and 100.

```
datainfo;
   in= employee.rxd out=outstats.dbf;
   options percentile_step=25;
run;
```

You can also use the `round` function to convert a variable with large numbers of values into a variable with a more manageable number of buckets. For example, a salary variable can be rounded to thousands to help you get an idea of the value distribution.

```
Salary = round(salary,1000);
```

The Output Database
The output database will have the following two variables added automatically:

`_stat_` text string with statistics name. The following is a list the strings and their definitions

- Records number of records read
- Count number of records with data
- Missings number of records with missing data
- Mean average value
- Sum total of all non-missing values
- Range Maximum minus minimum
- Std Dev Standard Deviation
- Std Err Standard Error
Min-#  5 lowest values (not 5 lowest records)
Max-#  5 highest values (not 5 highest records)
Mode   Value with most number of hits (if tie the lowest value)
Percentile Requested percentiles
Value-# Unique values sorted lowest to highest. Preceeded by count

More Values? If more values exist than the limit this will have a yes.

_index_ increment for the value# and percentile#

For each input variable you will get two output variables. The counts will be written as the variables with a c_ prefix.
Sort Module

Purpose
The Sort Module allows users to restructure the order of the output records by specifying a sort order. Beyond the by statement, the module is exactly the same as the compute module. Your sort module paragraphs can use the DBMS/COPY Plus record filtering, expression processing, function library, and condition computation while transferring and sorting records from the input database to the output database.

Syntax

```
sort;
in=inbase  out=outbase;
by variablelist;
statements;
run;
```

Syntax Elements

```
sort;  The first statement for all paragraphs must be the module name followed by a semicolon; in this case, sort;  .
```

```
by variablelist;
The keyword by can be followed by any number of variables. Each variable can be preceded by the keyword descending if the variable should have its largest value first. The default sort order is ascending – lowest value first.
```

Usage

In the following example (modified from the compute module documentation), the records in the output database will be ordered from the lowest number of years to the greatest number of years.

```
sort;
in= employee.rxd  out= employee.dbf;
by years;
if years >= 10 then
tenured = 't';
else
We can modify the above example to sort the data by ascending years but then within each year we can put the tenured records first. To get the “t” to appear before the “f” we will use the desending keyword.

```plaintext
tenured = 'f';
run;

sort;
in= employee.rxd out= employee.dbf;
by years descending tenured;
if years >= 10 then
tenured = 't';
else
tenured = 'f';
run;
```
General Statements

General statements are commands that can be used in the Compute, Datainfo and Sort Modules within DBMS/COPY Plus. Note: these statements plus many more are available in DBMS/Analyst.

The Assignment, Delete, If, Select and Stop statements are also available for use in the DBMS/COPY Interactive Equation Builder.

**Assignment** Computes new variables. Available for use in DBMS/COPY Interactive.

**Createformatvars** Controls the generation of formatted value variables.

**Database Specification** Specifies the input and output database files.

**Delete** Prevents selected records from being written to the output database system. Available for use in DBMS/COPY Interactive.

**Drop** Prevents selected variables from being written to the output database.

**Format** Specifies a format for a variable.

**If / Then** Conditionally executes statements. Available for use in DBMS/COPY Interactive.

**Inodbc** Input database specification when using the ODBC access method. (Includes Microsoft Access.)

**Keep** Identifies selected variables that will be written to the output database.

**Label** Associates a long descriptive string with a variable name.

**Modify** Modifies the length of character variables when they are written to the output database.

**Obs** Limits processing to a specified number of records.
**Order**
Places the variables in the output database in the specified order.

**Outodbc**
Output database specification when using the ODBC access method. (Includes Microsoft Access.)

**Rename**
Renames variables when they are written to the output database.

**Retain**
Used with the `_retain` option for user control of variable retention.

**Run**
Causes the current module to process the commands.

**Select**
Identifies selected records that will be written to the output database. Available in **DBMS/COPY Interactive**.

**Stop**
Stop the current module execution.

**Values**
Creates value lists for SAS, SPSS for Windows and Stata.

**Vartype**
Specifies how the variable will be written to the output database.
Assignment Statement

**Purpose**
Stores the results of an expression (either numeric or character) in a variable. Available in DBMS/COPY Interactive.

**Syntax**

```
variable = expression;
```

**Syntax Elements**

- **=** The symbol used to assign the expression value to the variable.
- **Variable** The name of the variable where the evaluated expression will be stored.
- **Expression** Any numeric or character expression, including expressions containing other variables. Special Variables may also be included.

**Usage**
The Assignment Statement is used to store the results of an expression in a variable. The variable may already exist, in which case the value will be overwritten in the output database by the value of the expression.

If the variable does not exist before the assignment, the variable is given the type (character or numeric) of the expression. If the expression is a character expression, the variable will be assigned the same length as the expression.

There is no limit to the number of Assignment Statements that can be included in one program.

A variable must exist before it can be used in an expression on the right side of an Assignment Statement. A variable exists if it is part of the input database (and not excluded by the use of a Drop or a Keep Statement) or if it has been created in a previous Assignment Statement.
Examples

\[ b = 'abcdef'; \]

\( b \) will become a character variable, and it will have a length of six characters.

\[ a = \log(\text{age}) \times 34; \]

\( a \) will become a numeric variable, provided \( \text{age} \) is previously defined as a numeric variable.

If / Then Statements can be combined with Assignment Statements to create a new variable based on the value of other variables. For example, in the employee database the person’s sex is stored as a single lowercase character. The new database will need the spelled out words \textit{Female} and \textit{Male}.

\[
\text{compute;}
\begin{align*}
\text{in} &= \text{employee.rxd} \quad \text{out} = \text{longval.db;}
\text{if} \quad \text{sex} &= 'f' \quad \text{then}
\quad \text{long}_\text{sex} &= '\text{Female}';
\text{else}
\quad \text{long}_\text{sex} &= '\text{Male}';
\text{label} \quad \text{long}_\text{sex} &= \text{Sex of Employee;}
\text{run;}
\end{align*}
\]

The first use of a variable determines its type and its length. In the above example, \textit{long}\_\textit{sex} is a character string of length 6. If the assignment of \textit{Male} had occurred before the \textit{Female}, the character string would be of length 4 and the Female would be shortened to \textit{Fema}. To avoid this, use the longest value first, or pad the first value with blanks to the length of the longest one.

When combining Assignment Statements with record filtering (Select or Delete Statements), variables that are needed for the filtering statement should go before the filtering statement. The other variables should go after the filtering statement. This will cause the modules to run faster.
Note:
If the name of the variable is in or out, surround the variable name with caret marks, (^), when using it in an Assignment Statement. For example

\[ ^{\text{in}} = \text{age} \times 12; \]

will assign the value of the age variable times 12 to the in variable. If the carets are omitted, DBMS/COPY Plus will consider the in = to be part of a Database Specification Statement, rather than an Assignment Statement.
Createformatvars Statement

**Purpose**
 Specifies how variables with value labels will be processed.

**Syntax**

Createformatvars prefix= suffix= files=;

**Syntax Elements**

- **Prefix=** Names the string that will be the prefix when creating the new variables containing the formatted values.

- **Suffix=** Names the string that will be the suffix when creating the new variables containing the formatted values.

- **Files=** List of SAS format libraries to search for the formats. After the equal sign, list the files in quotes.

**Usage**

DBMS/COPY can create new variables that will contain the formatted values of each variable that has a value label list. The name of the new variable will be the name of the raw variable with a prefix and/or suffix. The new variable will be a character variable and the length will be sufficient to store the longest value label. You only need to specify a prefix or suffix if you want to create the new formatted value containing variables. If you are moving data out of SAS and into SPSS or SAS and you want value lists created, you need to specify the format library files. If you are moving data out of SPSS or Stata and you don’t want the new formatted value variables, you don’t need this statement.
Examples

If two variables, jobcode and salarycode have formats and if you specify f_ as a prefix then two new variables f_jobcode and f_salarycode will be created. The syntax would look like:

```
createformatvars prefix=f_;
```

If you want to take a SAS file with user defined formats and write a SPSS file with value lists, you need to list the files but don't need a prefix or suffix.

```
createformatvars files="c:\temp\abc.sd2" "c:\temp\def.sd2";
```
Database Specification Statement

Purpose
The Database Specification Statement informs DBMS/COPY Plus which database files will be read from and to. It consists of two components, \textit{in=} and \textit{out=}.

Syntax
\texttt{in=filename.ext (options) out=filename.ext (options);}

Syntax Elements
- \textit{in=} Informs DBMS/COPY Plus the next file name is the input database.
- \textit{out=} Informs DBMS/COPY Plus the next file name is the output database.
- \textit{filename.ext} The name of the database file that will be used in the module. The extension must be the pseudo extension associated with the particular software system the file is written in.
  
The online help system has a listing of all supported packages and their pseudo extensions.
- \textit{(options)} The optional modifiers that control how DBMS/COPY Plus reads the input file and writes the output file. Each option must be enclosed in parentheses. There are Database options and Control options.

Reading and Writing ODBC databases
To read ODBC databases within DBMS/COPY Plus, use the \texttt{inodbc} statement. To write ODBC databases within DBMS/COPY Plus, use the \texttt{outodbc} statement.
Input Database Options

All the input database options effect the way DBMS/COPY Plus reads the input file. None of the options will make any changes at all to the input file. It is more efficient to drop= and keep= as input options rather than output options whenever possible. As input options they prevent DBMS/COPY Plus from even reading the unnecessary information and thus save processing time and space.

In the discussions below, the term variablelist refers to a list of variable names. There can be any number of variable names in the list. The names should be separated only by spaces.

Each input database option used must be in a separate set of parentheses.

(keep = variablelist)

The variables specified will be the only variables read from the input database. This is useful when the input database has many variables and only a few are needed. This option can speed up processing. The xyz10-xyz20 shorthand can be used.

(drop = variablelist)

The variables specified will be the only variables not read from the input database. This is useful when the database has many variables and only a few are not needed. This option can speed up processing. The xyz10-xyz20 shorthand can be used.

(rename = oldname=newname oldname=newname ...)

The rename= is followed by a list of oldname=newname pairs. The oldname is a name from the database. The newname is the name that will be used by the module. The rename option can be used with the keep= or drop= options.

(modify = variable-length variable-length ...)

The modify= option is used to change the length of character string variables when the module reads or writes the database. The modify= is followed by a list of variable-length pairs. The variable is followed by a dash and then the length that you want the module to think the variable is. If the length is longer than the actual database variable, it will be padded with blanks. If the length is
shorter than the actual database variable, it will be truncated.

(label variable=string;)  
The label option is used to give a variable a variable label. Labels will be used on printouts whenever the variable name would have been printed. The label is followed by a variable from the database, an equal sign, and label string. The string must be terminated by a semicolon. There can be any number of label options.

(keyfile = file)  
The keyfile option is used to specify a file where the database keys or index can be found. This is needed for systems where the main database does not indicate where the keys are.

**Input Control Options**

Any input control options used may be in one set of parentheses or they may be in separate sets.

(key=keynum)  
For a keyed database that stores multiple keys in one database this option lets you specify which key to use. The value of keynum is the number of key to use. The default is key=1. (This option applies to PRODAS databases only.)

(buffer=bufsize)  
**DBMS/COPY Plus** reads records from a database in multiple record chunks. The bigger the chunk, the fewer the actual disk accesses required and the faster the program runs. The drawback is that the bigger the buffer the less space is available for your program. The maximum buffer size is 65000. The default is 10000 bytes.

(obs=count)  
Change the number of records that will be read from the database. Normally the entire database is read. To speed up testing of a program you may want to set obs= to a small number.

(raw)  
The raw option is used with a keyed database to indicate the records can be read in any order. This option will read
the database is the fastest possible way. (This option applies to PRODAS databases only.)

**Output Database Options**

In the discussions below, the term *variablelist* refers to a list of variable names. There can be any number of variable names in the list. The names should be separated only by spaces.

Each output database option used must be in a separate set of parentheses.

*(keep = variablelist)*

The *keep* is followed by a list of variables. The variables specified will be the only variables written to the output database. This is useful when only a few variables are needed. This option will speed up processing. The *xyz10-xyz20* shorthand can be used.

*(drop = variablelist)*

The *drop* is followed by a list of variables. The variables specified will be the only variables not written to the output database. This is useful when only a few variables are not needed. This option will speed up processing. The *xyz10-xyz20* shorthand can be used.

*(rename = oldname=newname oldname=newname ...)*

The *rename* is followed by a list of *oldname=newname* pairs. The *oldname* is the name used during module execution. The *newname* is the name that will be written to the output database. The *rename* option can be used with the *keep* or *drop* options.

*(modify = variable-length variable-length ...)*

The *modify* option is used to change the length of character string variables when the module writes to the output database. The *modify* is followed by a list of *variable-length* pairs. The *variable* is followed by a *dash* and then the *length* that you want the output database to think the variable is. If the length specified is longer than the actual variable, the output variable will be padded with blanks. If the length is shorter than the actual variable, the output variable will be truncated.
The `label` option is used to store a variable label on the output database. Note: Some database systems do not have a place to store a label. The `label` is followed by a `variable` from the database, an equal sign, and `label string`. The string is terminated by a `semicolon`. There can be any number of label options, up to the number of variables.

The `vartype= variablelist typespec variablelist typespec ...;` option is used to change the variable type of a variable. Either the Vartype Statement or the `vartype= output option may be used in one DBMS/COPY Plus paragraph but not both. See the information on the Vartype Statement in the Compute Module Reference section for details on the available variable types.

The `dbslabel = string;` option is used to give a label to the database. This option is used to add a description to a database. (Note: not all packages support database labels)

The `key` option causes the module to generate a keyed database with the key consisting of the variables that follow the word `key`. Each variable can be preceded by the word `descending` if the variable should be in descending order.

**Output Control Options**

Any output control options used may be in one set of parentheses or they may be in separate sets.

The `buffer=buffsize` option is used to change the variable type of a variable. Either the Vartype Statement or the `vartype= output option may be used in one DBMS/COPY Plus paragraph but not both. See the information on the Vartype Statement in the Compute Module Reference section for details on the available variable types.

DBMS/COPY Plus writes records from a database in multiple record chunks. The bigger the chunk the fewer actual disk accesses and therefore programs run faster. The drawback is that the bigger the buffer the less space is available for your program. The maximum buffer size is 65000. The default is 10000 bytes.
(copykey) For database systems that can write a keyed database, the copykey option tells the module to copy any keys from the input database. (This option applies to PRODAS databases only.)

(delkey = N) For database systems that can write a keyed database, the delkey option tells the module to ignore the key with number N from the input database. If more than one key is to be ignored, list the key numbers separated by spaces (i.e. (delkey = 2 4 5)). (This option applies to PRODAS databases only.)

Examples
compute;
in=fit.spss (keep=age weight runtime runpulse)
    (rename = age=pat_age)
    (buffer=32000)
out=age_time.spss
    (label age=Patient's Age;)
    (label runtime=Time for One Mile;);
run;
Delete Statement

Purpose
The Delete Statement allows the module to control which records are passed to the output database. Available in DBMS/COPY Interactive.

Syntax

```
delete expression;
```

Syntax Elements

- **delete**: The keyword that tells DBMS/COPY Plus not to copy certain records to the output database.

- **Expression**: For each record in a database, the expression is evaluated. If true, the record is not passed to the output system. If the expression evaluates to false, the record is passed to the output system. There is a complete discussion of how expressions are evaluated in the *Expressions* chapter.

Usage

For each record in the input database, the expression is evaluated. If false (equal to 0), the record is passed to the output database. If the expression evaluates to true, the record is skipped.

The Select Statement performs a very similar action as the Delete Statement. The difference is that if the expression evaluates true for the record, the record is used. There can be only one Select or Delete Statement per module execution.

**DBMS/COPY Plus** provides both statements because sometimes tasks are more easily thought of as eliminating records instead of selecting records. Either statement can be used.

Examples

To create a smaller database from the fitness database containing only the records of people over 50:
The personnel department wants a database of employees excluding accounting and personnel.

```
compute;
  in= fit.spss  out= over50.spss;
  delete age < 50;
run;
```

The following Select Statement is equivalent. It illustrates the relationship between the Delete Statement and the Select Statement.

```
compute;
  in= employee.rxd  out= notadmin.rxd;
  delete depart='accounting' or depart='personnel';
run;
```

```
compute;
  in= employed.dbf out=temp.mtw;
  select depart<>'accounting' and depart<>'personnel';
run;
```
Drop Statement

Purpose
Specifies a list of variables to exclude from the output database.

Syntax
   drop variablelist;

Syntax Elements
Drop  The keyword for this command
Variablelist  The variables that will not be written to the output database. The variable name shorthand notation xyz12-xyz23 is available.

Usage
All the variables in the input database and all variables created during the processing with Assignment Statements will be written to the output database, except for those variables listed in the Drop Statement. The Drop Statement is helpful if you have created new variables to use in the internal processing, perhaps to use in If / Then Statements, that will not be needed in the output file. It is also useful if there are more variables in the input database than are needed in the output database.

The Drop Statement has the same effect as the drop= option for the output database. The Keep Statement is very similar to the Drop Statement. The difference is that the Keep Statement specifies the variables that will be written to the output database.

One paragraph can have only one Drop, Order, or Keep Statement. Also, if there is a drop= or keep= output database option, the drop= or keep= option will overwrite the Drop statement. A drop= or keep= input database option will have no effect on the Drop statement.

Examples
compute;
   in=fitness.db out=outdbs.dbf;
drop a b;
run;

compute;
in=fitness.db (drop= a b)  out=outdbs.dbf;
run;

The above two Compute Modules have identical output files. The second form (drop with the input database) is recommended. Since the a and b variables are not needed in the paragraph execution, there is no need for DBMS/COPY Plus to read them at all.
Format Statement

Purpose
Used to store a format specification with a variable.

Syntax A
format formatspec variablelist;

Syntax B
format variablelist formatspec variablelist formatspec ...;

Syntax Elements
Format  The keyword for this statement.
Formatspec The format that will be associated with the variables.
Variablelist A list of variables that should be assigned the format. The variable name shorthand notation abc12-abc18 is acceptable.

Usage
The first form of the statement (Syntax A) requires the format specification first and then the variables that should be assigned that format. Only one format per statement is allowed.

The second form of the statement (Syntax B) requires the variables first and then the format for the preceding variables. The variable-format groups can be repeated any number of times.

The format will be stored with the variable and will be carried to any created databases.

To erase a format use the formatspec of only a period.
The tables below describe the **DBMS/COPY** variable formats. In the tables, the letters *N* and *M* represent numbers. The period is a required part of all **DBMS/COPY Plus** variable formats.

### Numeric Formats

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fN.M</td>
<td>Numeric outputs will be printed in N columns. There will be M decimal digits. No commas will be inserted. If M is not present, or zero, the decimal point will not print.</td>
</tr>
<tr>
<td>fixedN.M</td>
<td></td>
</tr>
<tr>
<td>N.M,</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>123.67 ( f10.3 ) \rightarrow 123.670</td>
</tr>
<tr>
<td>CommaN.M</td>
<td>Numeric outputs will be printed in N columns. There will be M decimal digits. Commas will be inserted at the usual points, following thousands, millions and billions. If M is not present, or zero, the decimal point will not print.</td>
</tr>
<tr>
<td>Example</td>
<td>23456.78 ( comma9.2 ) \rightarrow 23,456.78</td>
</tr>
<tr>
<td>DollarN.M</td>
<td>Numeric outputs will be printed in N columns. There will be M decimal digits. Commas will be inserted at the usual points, preceding thousands, millions and billions. If M is not present, or zero, the decimal point will not print. The first blank on the left-hand side will be replaced by a dollar sign.</td>
</tr>
<tr>
<td>Example</td>
<td>23456.78 ( dollar10.2 ) \rightarrow $23,456.78</td>
</tr>
<tr>
<td>AsterN.M</td>
<td>Numeric outputs will be printed in N columns. There will be M decimal digits. Commas will be inserted at the usual points, following thousands, millions and billions. If M is not present, or zero, the decimal point will not print. The blanks on the left-hand side will be filled with asterisks (*).</td>
</tr>
<tr>
<td>Example</td>
<td>1234.5 ( aster9.1 ) \rightarrow **1,234.5</td>
</tr>
<tr>
<td>BestN.</td>
<td>Numeric outputs will be printed in N columns. The program module makes a determination as to how data should be printed, and will attempt to print the entire number. If the number is too large for the number of columns available, the module will round the number or,</td>
</tr>
</tbody>
</table>
failing that, will print the number in exponential form.

Examples, 12345678901 \( \text{best10.} \rightarrow 1.2345e+10 \)
1234 \( \text{best10} \rightarrow 1234 \)

\( eN. \) Numeric outputs will be printed in \( N \) columns. The number will be printed in exponential form. The minimum number of columns is 7. Beyond 7 columns, the number of decimal digits is increased. The number is printed with one digit before the decimal point.

Example, 12345678901 \( e10. \rightarrow 1.2345e+10 \)

---

### Date Formats

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateN.</td>
<td>The date will be written right-justified in ( N ) columns. The date will be displayed in a ddMmmyy format, dd representing the day of the month, Mmm the 3-character abbreviation for the month and yy the last 2 digits of the year.</td>
</tr>
<tr>
<td></td>
<td>Example, date15. 12Feb83</td>
</tr>
<tr>
<td>DatehN.</td>
<td>The date will be written in ddMmmyy:hh format, with hh representing the number of hours in the day based on a 24-hour clock. The hours are always printed in two columns,</td>
</tr>
<tr>
<td></td>
<td>Example, dateh15. 12Feb83:20</td>
</tr>
<tr>
<td>DatehmN.</td>
<td>The date will be written in ddMmmyyhh:mm format, with mm representing the number of minutes in the hour. The minutes will always be printed in two columns.</td>
</tr>
<tr>
<td></td>
<td>Example, datehm15. 12Feb83:20.45</td>
</tr>
<tr>
<td>MdyN.</td>
<td>The date will be written right-justified in ( N ) columns. The date will be displayed in mm/dd/yy format, with mm representing the month, dd the day of the month and yy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format Statement</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdy16.</td>
<td>The date will be written in mm/dd/yy format, with the year. (If only 6 or 7 columns are specified for the date, the slashes will not be printed.),</td>
<td>02/12/83</td>
</tr>
<tr>
<td>MdyhN.</td>
<td>The date will be written in mm/dd/yy:hh format, with hh representing the number of hours elapsed on a 24-hour clock.</td>
<td>Mdyh16. 02/12/83:20.45</td>
</tr>
<tr>
<td>DmyN.</td>
<td>The date will be written right-justified in N columns. The date will be written in dd/mm/yy format, with dd representing the day of the month, mm the month and yy the year.(If only 6 or 7 columns are specified for the date, the slashes will not be printed.),</td>
<td>Dmy16. 12/02/83</td>
</tr>
<tr>
<td>dmyhN.</td>
<td>The date will be written in dd/mm/yy:hh format, with hh representing the number of hours elapsed on a 24-hour clock.,</td>
<td>dmyh16. 12/02/83:20</td>
</tr>
<tr>
<td>dmyhmN.</td>
<td>The date will be written in dd/mm/yy:hh.mm format, with mm representing the number of minutes in the hour.</td>
<td>dmyhm16. 12/02/83:20.45</td>
</tr>
<tr>
<td>mmddN.</td>
<td>The date will be written in mm/dd format, with mm representing the month and dd the day of the month.</td>
<td>mmdd8. 02/12</td>
</tr>
<tr>
<td>mmyyN.</td>
<td>The date will be written in mm/yy format, with mm representing the month and yy the year.,</td>
<td>mmyy8. 02/83</td>
</tr>
<tr>
<td>yyN.</td>
<td>The date will be written in yy format, with yy representing the year.</td>
<td>yy8. 83</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>yyyyN.</td>
<td>The date will be written in yyyy format, with yyyy representing the year.</td>
<td>yyyy8. 1983</td>
</tr>
<tr>
<td>ddN.</td>
<td>The date will be written in dd format, dd representing the day of the month.</td>
<td>dd8. 12</td>
</tr>
<tr>
<td>mmN.</td>
<td>The date will be written in mm format, mm representing the month.</td>
<td>mm8. 2</td>
</tr>
<tr>
<td>DdmmN.</td>
<td>The date will be written in dd/mm format, with dd representing the day of the month and mm representing the month.</td>
<td>ddmm8. 12/2</td>
</tr>
<tr>
<td>DdmmmN.</td>
<td>The date will be written in ddmmm format, with dd representing the day of the month and mmm representing the month.</td>
<td>ddmmm8. 12feb</td>
</tr>
<tr>
<td>MmmyyN.</td>
<td>The date will be written in mmmyy format, with mmm representing the month and yy the year.</td>
<td>mmmyy8. feb83</td>
</tr>
<tr>
<td>mmmN.</td>
<td>The date will be written in mmm format, with mmm representing the month.</td>
<td>mmm8. feb</td>
</tr>
</tbody>
</table>

**Character Formats**

$N.$

The $ indicates to **DBMS/COPY Plus** this is a character format. $N.$ represents the length. The only format **DBMS/COPY Plus** associates with character variables is the length.
Normally a character format is not used since the default is to use all of the characters in the string. This format can be used to pad a variable with blanks by specifying a format value longer than the variable. This format will not truncate a character variable. Use the `substr` or the `subcol` function to select only a part of a character variable.

**Example**

To assign the format e12. to the variables `plank` and `amount`:

```plaintext
format e12. plank amount;
```
If / Then Statement

Purpose
The If / Then Statement controls the execution of other statements. The If / Then Statement can be used to conditionally compute new variables or change the value of existing variables. Available for DBMS/COPY Interactive.

Syntax A
if expression then
  statement;

Syntax B
if expression then
  statementA;
else
  statementB;

Syntax Elements
if  then      Required keywords for the If / Then Statement.
Else          Optional keyword for the If / Then Statement.
Expression    Any expression that can be evaluated as either true or false. See the Expressions chapter of the manual for further information.
Statement     A complete Assignment, Delete, or Select Statement or another If / Then Statement. The statement can be enclosed within a Begin / End or a Do / End block.

Usage
Either Syntax A or Syntax B is acceptable.

The expression is evaluated to determine if it is true or false.
The If / Then Statement executes the statement after the Then keyword when the expression is true. If Syntax B is used, the If / Then Statement executes the statement after the Else keyword when the expression is false.

When it is necessary to execute more than one statement for the true or false condition, the statements must be surrounded by a Begin / End block or a Do / End block. These two blocks are equivalent.

If / Then Statements can be nested inside other If / Then Statements. The Else Statement, if any, will apply to the immediately previous If. An If Statement contained in a different Begin / End or Do / End block from the Else is not considered to be a “previous If”.

If Statements can be nested to any level.

If there are nested If / Then Statements, particularly if there are Else Statements, the use of Begin / End blocks or Do / End blocks is highly recommended.

**Examples**

```plaintext
compute;
    in= fit.spss   out= fitgroups.db;
    if sex = 'f' then
        begin;
            if age < 50 then
                group = 1;
            else
                group = 2;
        end;
    else
        begin;
            if age < 50 then
                group = 3;
            else
                group = 4;
        end;
    end;
run;
```

This **DBMS/COPY Plus** paragraph is an example of nested If / Then Statements with Begin / End Blocks. The result of the paragraph is a Paradox file called `fitgroups.db`. There will be an additional variable called `group`. Women aged less than 50 will be assigned `group 1`, women 50 and older will be assigned `group 2`, men under 50 will be assigned `group 3`, and men over 50 will be assigned `group 4`. 
**Begin – End and Do -- End Blocks**

**Purpose**
These blocks are used in the If / Then Statement when more than one statement is needed to be executed for a true or false condition. Begin / End Blocks and Do / End Blocks are equivalent.

**Syntax For Begin / End**
```
Begin;
   StatementA;
   StatementB;
   StatementC;
End;
```

**Syntax For Do / End**
```
Do;
   StatementA;
   StatementB;
   StatementC;
End;
```

**Usage**
**DBMS/COPY Plus** treats the entire block as if it were one statement. Semicolons are required at the end of each statement in the block.
Inodbc Statement

Specifies the ODBC source and the SQL select statement that will be viewed by **DBMS/COPY Plus for Windows** as the input database. The `inodbc` statement is only available in DBMS/COPY Plus for Windows.

**Syntax**

```
inodbc [source] select_statement;
```

**Syntax Elements**

- **Inodbc** The keyword for this statement.
- **[source]** The odbc source is specified inside the brackets. The source can contain a large number of parameters. We recommend running the transfer interactively once to see what parameters are used.
- **select_statement** The text of the SQL select statement that DBMS/COPY should use as input. For example, to retrieve the variables a, b, c from the xyz database:

  ```
  select a, b, c from xyz
  ```

**Usage**

The `inodbc` statement replaces the `in=` database specification. To specify the output database use an `out=` or `outodbc` database specification statement. For example, to select the variables a,b,c from the xyz database and write them to a SAS for Windows Version 7 file:

```
compute;
inodbc [DSN=QEDBF] select a,b,c from xyz.dbf;
out=threev.sd7;
run;
```

Please notice the semicolon at the end of the `inodbc` statement and the separate `out=` statement. In the above program, DBMS/COPY "sees" three variables from the input database. These variables can be used in statements and expressions just as any other database variables. For example, to save the log of c to the output database.
compute;
inodbc [QEDBF] select a,b,c from xyz.dbf;
out=threev.sd7;
logc = log10(c);
run;

The above simple example shows a select statement with only three variables but the select statement can be any valid complex SQL select statement.

Notes
The DBMS/COPY Interactive for Windows interface has a built-in easy-to-use SQL select statement Query Builder. The Interactive system can save the complete compute module code (including the inodbc statement) to a disk file. We highly recommend using the Interactive system to create and save the code. Seeing the exact code makes a great starting point from which changes can be made and rerun.
Keep Statement

*Purpose*
Specifies a list of variables to include in the output database.

*Syntax*
```plaintext
keep variablelist;
```

*Syntax Elements*
- **Keep** The keyword for this command.
- **Variablelist** The variables that will be written to the output database. The variable name shorthand notation `xyz12-xyz23` is available.

*Usage*
Only those variables that are listed in the Keep Statement will be written to the output database. The Keep Statement is helpful if you have created new variables to use in the internal processing, perhaps to use in If / Then Statements, that will not be needed in the output file. It is also useful if there are more variables in the input database than are needed in the output database.

The Keep Statement has the same effect as the `keep=` option for the output database. The Keep Statement is very similar to the Drop Statement. The difference is that the Drop Statement specifies the variables that will not be written to the output database. One way to choose between the Drop and Keep Statements is which will require the fewer variables names to be typed. If there are thirty variables and only three are needed, the Keep Statement is more appropriate. If there are thirty variables and twenty five are needed in the output database, the Drop Statement is more efficient.

The Order Statement is very similar to the Keep Statement. The Order Statement also lists the variables that will be written to the output database. The difference between the statements is that the Keep Statement preserves the existing order of the variables and the Order Statement changes the order of the variables.
One paragraph can have only one Drop, Order, or Keep Statement. Also, if there is a \textit{drop=} or \textit{keep=} output database option, the \textit{drop=} or \textit{keep=} option will overwrite the Keep Statement. A \textit{drop=} or \textit{keep=} input database option will have no effect on the Keep Statement.

\textbf{Examples}

\begin{verbatim}
compute;
  in=fit.spss  out=outdbs.dbf;
  keep sex age;
  run;

compute;
  in=fit.spss (keep= sex age)
  out=outdbs.dbf ;
  run;
\end{verbatim}

The above two Compute Modules have identical output files. The second form (keep with the input database) is recommended. Since the \textit{sex} and \textit{age} variables are the only variables needed in the paragraph execution, there is no need for \textbf{DBMS/COPY Plus} to read any of the other variables at all.

The result of either module execution will be a dBase file with two variables, \textit{sex} and \textit{age}.
Label Statement

Purpose
Assigns a long description to variable.

Syntax

```plaintext
label variable = string;
```

Syntax Elements

- **Label**  The keyword for this statement.
- **Variable**  The name of the variable to which the label is assigned.
- **String**  String that will be the variable's label. The string ends with a semicolon. If you need a semicolon in the string, surround the string with single quotes. If you need both semicolons and quotes in the string, for inside quotes use two quotes.

Usage
In the Compute Module, if the output database system supports variable labels, the labels will be written to the output file. Labels are not added to input databases. If you want to eliminate a variable's label, set it to a null string by placing a semicolon immediately after the equal sign.

The label statement is very similar to the `label=` option in the Database Specification Statement.

Examples

```plaintext
label sex = Subject's Sex;
label r_rate = Residual Heart Rate;
label withsemicolon = 'This Label has a ; in it';
label semiquote = 'This has both '' and ; in it';
label var_name=;
```
Modify Statement

Purpose
The Modify Statement is used to change the length of character variables. The variable's length will be modified when the variable is written to the output database. The variable's length will not be changed during the current module execution.

Syntax
modify variable-length;

Syntax Elements
modify The keyword for this statement.
variable The name of the character variable whose length is to be changed.
- The separator used in this statement is a hyphen.
Length The new length of the character variable. This must be a positive integer.

Usage
There can be any number of variable-length combinations.

This statement is overwritten by the (modify variables) option on the out= statement. The Modify Statement is used for databases that do not have a modify option on output databases.

If the length specified is longer than the actual variable, the output variable will be padded with blanks. If the length is shorter than the actual variable, the output variable will be truncated.
Examples
The Modify Statement to write the variable \textit{abc} as length 10 and \textit{xyz} as length 20 is

\begin{verbatim}
modify abc-10 xyz-20;
\end{verbatim}

A \textbf{DBMS/COPY Plus} paragraph to write the character variable \textit{sex} with length 8 is

\begin{verbatim}
compute;
  in=fit.spss  out=out.spss;
  modify sex-8;
run;
\end{verbatim}

The resulting output file will be identical to the input file, except the variable \textit{sex} will have a length of 8.
Obs Statement

Purpose
Informs Causes the module to pretend the input database has a specified number of records.

Syntax
\[ \text{obs } N; \]

Syntax Elements
- **Obs**: The keyword for this statement.
- **N**: The number of records for the module to use. N must be a positive integer.

Usage
This statement is very good for testing a module with a few records before taking the time to run the entire database.

If a larger number is specified than there are records in the input database, only the number of records in the database will be used.

Do not use an equal sign as a part of this statement. If you do, **DBMS/COPY Plus** will treat the statement as an Assignment Statement. A new variable called *obs* will be added to the output database and it will have the value of N for each record.

Example
To test a Compute Module with the first 50 records of a database.

```plaintext
compute;
in=bigdbs.dbf out=smalldbs.dbf;
obs 50;
run;
```
Order Statement

**Purpose**
Specifies a list of variables to include in the output database and gives the order the variables will be written in.

**Syntax**
```
order variablelist;
```

**Syntax Elements**
- **Order** The keyword for this command.
- **Variablelist** The variables that will be written to the output database in the order they will be written. The variable name shorthand notation `xyz12-xyz23` is available.

**Usage**
The Order Statement allows the user to specify the order in which variables will be written to the output database. If there is an Order Statement, only the variables in the Order Statement will be written to the output database. Any variables that are not in the Order Statement will be dropped.

The Order Statement is very similar to the Keep Statement. The Keep Statement also lists the variables that will be written to the output database. The difference between the statements is that the Keep Statement preserves the existing order of the variables and the Order Statement changes the order of the variables.

One paragraph can have only one Drop, Order, or Keep Statement. Also, if there is a `drop=` or `keep=` output database option, the `drop=` or `keep=` option will overwrite the Order Statement. A `drop=` or `keep=` input database option will have no effect on the Order Statement.
Examples

compute;
    in=employee.rxd out=emp.spss;
    order jobtitle depart salary sex;
run;

This paragraph will create an SPSS/PC+ file with just four variables, *jobtitle*, *depart*, *salary*, and *sex*. The variables will be in the order listed.
Outodbc Statement

Specifies the ODBC source and the output table name that
DBMS/COPY Plus will use as the output database.

Syntax

outodbc [source] tablename append delete_first;

Syntax Elements

Outodbc The keyword for this statement.

[source] the odbc source is specified inside the brackets. The source can contain a large number of parameters. We recommend running the transfer interactively once to see what parameters are used.

Tablename the output tablenamne. For some packages (including dBase and Paradox) this could be a disk filename. For other packages this could be the name of a table within a large managed database system. The following saves the results to the dBase filec:\dbmscopy\files\xyz.dbf:

outodbc [QEDBF] c:\dbmscopy\files\xyz.dbf;

append DBMS/COPY can append records to an existing table. Put the keyword append after the output tablename if the data should be added to an existing table.

Outodbc[Access7...] masterdbs append;

delete_first For DBMS/COPY to overwrite a table, the table must first be deleted. To delete the table if it exists, put the keyword delete_first after the output tablename. Note: you can use this option even if the tablename doesn’t exist.

Outodbc[Access7...] outtablename delete_first;
**Usage**

The *outodbc* statement replaces the *out* database specification. To specify the input database use an *in* or *inodbc* database specification statement.

```
compute;
in=acctrec.rxd;
outodbc [QEDBF] d:\newdata\acctrec.dbf;
run;
```

Please notice the semicolon at the end of the *outodbc* statement and the separate *in* statement. Executions can read data from one odbc source and write to another. For example, to select the variables a,b,c from the xyz database and write them to an ODBC Paradox file:

```
compute;
inodbc [QEDBF] select a,b,c from xyz.dbf;
outodbc [PARODBC] threev.db;
run;
```

**Notes**

The DBMS/COPY Interactive for Windows can save the complete compute module code (including the outodbc statement) to a disk file. We highly recommend using the Interactive system to create and save the code. Seeing the exact code makes a great starting point from which changes can be made and rerun.
Rename Statement

Purpose
Rename variables when creating an output database.

Syntax
rename oldname=newname oldname=newname ...;

Syntax Elements
Rename The keyword for the Rename Statement.
Oldname The original name of the variable that should be stored in the output database under a different name.
Newname The new name that the variable should be stored under in the output database. This name is not used in the module. It only appears on output databases.

Usage
There can be any number of oldname=newname pairs.

The Rename Statement is used for output databases that do not have a rename= option on the output database. If you want to change the variable name that is used in the module execution, use the rename= input database option. The Rename Statement can then be used to return the original names to the variables for the output database if desired. The input database will not be changed in any way.
Examples

The following two programs are equivalent. The second form is recommended because output database options should appear with the database.

compute;
  in=fitness.spss out=outdbs.rxd;
  rename sex = patient_sex;
run;

compute;
  in=fitness.spss
    out=outdbs.rxd (rename = sex=patient_sex);
run;
Retain Statement

_Purpose_
Informs the Compute Module to retain the previous value of the variables listed in the Retain Statement. It is only used when the _retain_ option is set to _y_.

_Syntax_
```
retain variablelist;
```

_Syntax Elements_
- **Retain** The keyword for this statement.
- **Variablelist** List of variables used within the program that should be retained. The variable name shorthand notation `def3-def12` is acceptable. Variables must be defined before they can be used in a Retain Statement.

_Usage_
Normally, the module will automatically retain the previous value of the created variables independent of the particular record being processed. Some users, especially those familiar with the SAS Retain Statement, prefer to specify whether a created variable will retain its value or be reset to a missing value. The Retain Statement gives that power to the user. It is enabled by the _retain_ option. When _retain_ is set to _y_, all created variables are reset to missing values for each record unless assigned a new value for that record.

After the _retain_ option is set to _y_, the Retain Statement can be used to specify which variables will keep their values and not be reset to missing values.

For example, consider the following input file called `number.dbf`. 
If the _retain option is set to the default, n, the following paragraph will have the result given below.

```
compute;
  in=number.dbf  out= out1.dbf;
  if number < 30 then
    i = number;
  else
    j = number;
run;
```

Record Variables:

<table>
<thead>
<tr>
<th>Record</th>
<th>Number</th>
<th>I</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>15</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>25</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>25</td>
<td>45</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>
If _retain is set to y, then the same paragraph will have the following results

<table>
<thead>
<tr>
<th>Record</th>
<th>Variables:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>I</td>
<td>j</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td></td>
<td>50</td>
</tr>
</tbody>
</table>

In order to have the previous results, with i retaining the value of 25, use the same paragraph, but add a Retain Statement just before the Run Statement.

```plaintext
compute;
in=number.dbf out= out1.dbf;
if number < 30 then
  i = number;
else
  j = number;
retain i;
run;
```
Run Statement

Purpose and Usage
When you are in any module, the module expects commands until the Run Statement is given. The Run Statement tells the module that there are no more commands and to start processing.

Every module execution must begin with the module name and end with the Run Statement.

Syntax

```plaintext
module name;
statements;
run;
```

DBMS/COPY Plus Note
When running a program file from within DBMS/COPY, control will return after the program file is finished. When running a program file from the command line, control will return to the calling application. The calling application could be the Program Manager or some other program.
Select Statement

Purpose
The Select Statement allows the module to control which records are passed to the output database. Available in DBMS/COPY Interactive.

Syntax
```
select expression;
```

Syntax Elements
- `select` The keyword for the Select Statement.
- `expression` Any expression that evaluates to a numeric value.

Usage
For each record in the input database, the expression is evaluated. If true (not equal to 0) the record is passed to the output database. If the expression evaluates to false (0), the record is skipped.

For a discussion of what can make up an expression, please read the Expressions and Functions chapters.

The Delete Statement performs the same action in reverse as the Select Statement. If the expression evaluates true for the record, the record is skipped. There can be only one Select or Delete Statement per module execution.

Both select and delete statements are provided because sometimes tasks are more easily thought of as eliminating records instead of selecting records. Either statement can be used.
Examples

The personnel department wants to know which employees that earn over 40,000 dollars have a poor rating of 4 or lower. Since a missing numeric value is smaller than the smallest number, a missing value rating will be considered less than 4.

```
compute;
    in=employee.rxd out=badbucks.spss;
    select salary > 40000 and rating <= 4;
run;
```
Stop Statement

**Purpose**
The Stop Statement is an executable statement that ends the current module execution. Available in DBMS/COPY Plus and DBMS/COPY Interactive.

**Syntax**
```
stop;
```

**Usage**
For some programs, you might want to end the transfer once the records meet a specific criteria. For example, if you know a database is sorted by a date variable and only want the records for 1983. Why keep reading records after 1983?

```
compute;
  in=acctpay.db out=acctpay.sd2;
  if year(invdate) > 1983 then
    stop;
  select year(invdate);
run;
```
Values Statement

Purpose
The values statement is used to create value lists for writing to SPSS for Windows and Stata files.

The values statement creates a list of value labels for a variable.

Syntax
values varname #=value #=value ... ;

Syntax Elements
values keyword for the statement
varname name of the variable for which the value list will be assigned.
# raw number
value is the value label string.

The value needs to be in quotes (‘ or ”) if the string is more than a simple combination of alphanumerics.

The #=value group is repeated for each raw value you need mapped.

Usage
The statement can be used in the Interactive Mode's Equation Builder or in the batch mode. Each variable needs to have its own statement.

Example
values jobcode 0="no job" 1=boss
2='Boss of boss'
3=programmer 4=Vp;
Vartype Statement

Purpose
Different database systems store data in different formats. DBMS/COPY and DBMS/COPY Plus compensate for these different formats automatically. The DBMS/COPY manual provides further information about the ways that variables are translated. There may be instances when you want to specify the format or variable type yourself. The Vartype Statement allows the user to specify how numeric variables are written to the output database.

Syntax A
vartype typespec variablelist;

Syntax B
vartype variablelist typespec
variablelist typespec ...;

Syntax Elements
Vartype The keyword for the Vartype Statement.
Typespec The variable type that will be associated with the variables.
Variablelist A list of variables that should be assigned the variable type. The shorthand notation abc12-abc18 is acceptable.

Usage
The are two acceptable forms of the statement

Syntax A requires the type specification first and then the variables that should be assigned that type. Only one type per statement is allowed. Multiple Vartype Statements can be included in one module.

Syntax B requires the variables first and then the type for the preceding variables. The variable-type groups can be repeated any number of times.
All modules internally store two types of variables: character and numeric. The numeric values are internally stored as double precision (8 byte) floating point numbers. Dates are stored as the number of minutes since January 1, 1900.

The Vartype Statement controls how the internal numeric values will be written to the output database. For instance, it may be sufficient to store a variable as a two byte integer even though DBMS/COPY Plus would store it as an eight byte floating point number. Using the smaller vartype will result in a savings of time and memory space.

Either the Vartype Statement or the $vartype=\,$ option (from the Database Specification Statement) may be used in one module execution, but not both.

**Available Variable Types**

The following types are supported:

<table>
<thead>
<tr>
<th>Vartype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double.</td>
<td>double precision (8 byte) floating point numbers. The number will be accurate to 15 digits.</td>
</tr>
<tr>
<td>float.</td>
<td>single precision (4 byte) floating point numbers. The number will be accurate to 7 digits.</td>
</tr>
<tr>
<td>short.</td>
<td>Write the number as a two byte signed number. This supports -32768 to 32767.</td>
</tr>
<tr>
<td>unshort.</td>
<td>Write the number as a two byte unsigned number. This supports 0 to 65535.</td>
</tr>
<tr>
<td>long.</td>
<td>Write the number as a four byte signed number. This supports -2 billion to 2 billion.</td>
</tr>
<tr>
<td>unlong.</td>
<td>Write the number as a four byte unsigned number. This supports 0 to 4 billion.</td>
</tr>
<tr>
<td>byte.</td>
<td>Write the number as a one byte signed number. This supports -128 to 127.</td>
</tr>
<tr>
<td>unbyte.</td>
<td>Write the number as a one byte unsigned number. This supports 0 to 255.</td>
</tr>
</tbody>
</table>
sasnum.  SAS/PC stores floating point numbers using 3 to 8 bytes.  The sasnum. format accommodates them.

logic.,  Write the number as a true/false flag.

date.,  Write the internal representation of the date to the output system as a date variable.

time.  Write the internal representation of the time to the output database system as a time variable.

datetime.  Write the internal representation of the combination of the date and the time as a datetime variable.

fixedN.M  store numbers in a fixed form with a specific width and specific number of decimals.

The period is a required part of the variable typespec.

To erase a vartype use the typespec of only zero and a dot (0.).

**Examples**

```
compute;
  in=employee.rxd (drop= sex rating years reviewdate)
  out=sal1000.rxd;
  sal1000 = salary / 1000;
  vartype sal1000 short.;
run;
```

The new database will contain a variable called *sal1000*. *Sal1000* will contain the number of thousands that are in the *salary* variable. The *sex, rating, years, and reviewdate* variables will not be included. The *depart, salary, and jobtitle* variables will be included in the output database along with the new variable.
Expressions

Expressions are used by the If / Then Statements, Select Statements, Delete Statements and Assignment Statements. These statements are available in DBMS/COPY Interactive and DBMS/COPY Plus.

The free license to DBMS/Analyst increases the statements that can use expressions. Please see the DBMS/Analyst documentation for more information.

There are three types of expressions: numeric expressions, character expressions, and logical expressions. An expression is a combination of variable names, constants, operators and functions. If a variable name is part of an expression, then the value the variable contains is used to calculate the value of the expression.

In an If / Then Statement, the expression is used to determine if the statement following the Then keyword should be executed or if the statement following the Else keyword should be executed.

In a Select or Delete Statement, the expression is used to determine if the input record should be processed.

In an Assignment Statement, the expression is used to compute what should be assigned to the variable. For new variables the type of expression determines the new variable's type.

This chapter of the user's manual describes what can go into an expression. The way expressions can be used in statements is described in greater detail in the chapters in the manual on the individual statements.

This chapter covers the following topics:

Structure  How to build an expression from variables, constants, operators and parentheses.

Arithmetic Operators  How to use the add, subtract, multiply and divide operators.

Relational Operators  How to use the equality, less than, greater than and other relational operators.
Logical Operators
How to use the and, or, not logical operators.

Parentheses
Since expressions can get complex, parentheses give you control over the evaluation ordering.

Constants
DBMS/COPY supports character, numeric, and date constants. Constants are fixed items that do not vary from record to record.

Variables
DBMS/COPY can process variables names from the simple xyz to the complex Patient's Address.

Special Variables
DBMS/COPY maintains automatic special variables that you can use. These include today's date, the current record number, and π.

Missing Values
DBMS/COPY can process missing or null values. Missing values are used when the value is not known.

Concatenation
Concatenation is combining character strings into one string. For example, firstname and lastname combined into one variable called fullname.

Reserved Words
DBMS/COPY has no reserved words.

Functions
DBMS/COPY provides a large array of functions that can be included in expressions. The functions include math, trigonometry, probability, financial, date, and character functions. Information on the available functions is contained in a separate chapter.

Structure
An expression consists of a combination of constants, variables, arithmetic operators, relational operators, logical operators and functions.
For example,

\[ a + 4 \times b / c \]
\[ 'abc' \ <> \ y \]

are both expressions. In the following pages, the term *expression* implies any valid combination of operators, functions, variables and constants. For example, the definition of the *floor* function requires a numeric expression inside the parentheses. Therefore the following are both valid, provided the variable \( a \) contains a numeric value:

\[ \text{floor}(a \times 4) \]
\[ \text{floor}(\log(a/4)) \]

In *DBMS/COPY*, operators are symbols that perform specified procedures or operations. There are three types of operators, arithmetic, relational, and logical. Negation and the logical *not* are performed first. Arithmetic operators are performed before relational and logical operators. Any number of operators can be combined with other elements in an expression. The types of operators are summarized below.

**Arithmetic**

+ addition
- subtraction
* multiply
/ division
\text{mod} modulus
- negation

**Relational**

= equal
\(<> \) not equal
< less than
\(<= \) less than or equal to
> greater than
\(>= \) greater than or equal to

**Logical**

and Boolean and
or Boolean or
not Boolean not
Arithmetic Operators

Arithmetic operators perform addition, subtraction, division, multiplication, modulus, and negation.

+  Addition  Adds two numbers together

-  Subtraction  Subtracts the number on the right side of the symbol from the number on the left side.

*  Multiplication  Multiplies two numbers together.

/  Division  Divides the number on the right side of the symbol into the number on the left side.

mod  Modulus  Divides the number on the right side of the symbol into the number on the left side and returns the remainder. The modulus symbol is the word \textit{mod}. The modulus of 5 and 3 (5 \text{mod} 3) is 2. The modulus is also available as a function. The syntax of the function is \texttt{mod(x,y)}. This is equivalent to $x \text{ mod } y$.

-  Negation  Negation reverses the sign of a number. Negation is an unary operator and is performed before any other operators except the logical \textit{not}.

Operation Order

- Negation is performed first.
- Multiplication, division, and modulus are performed left to right.
- Addition and subtraction are then performed left to right.

Relational Operators

Relational operators compare two values for equality, inequality, less than, less than or equal to, greater than, greater than or equal to.

Relational operators can compare two numeric values or two character values.

The result of the comparison is:

\[
\begin{align*}
1 & \text{ if the comparison is } \text{true.} \\
0 & \text{ if the comparison is } \text{false.}
\end{align*}
\]
Relational operators are performed after arithmetic operators.

The symbols for the relational operators are:

- `=` equality
- `<>` inequality
- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to

**Combining Operators**

Arithmetic and relational operators can be combined in one equation. For example:

\[
a + 3 = b \times c
\]

is evaluated as:

- 1. The value of the variable \(a\) and the numeric constant 3 are added and stored in an internal variable.
- 2. The values of the variable \(b\) and the variable \(c\) are multiplied together and stored in an internal variable.
- 3. The first internal variable is compared to the second internal variable for equality.
- 4. If the result of step 3 is true, the expression is given the value of 1; otherwise the expression is given the value of 0.

**Comparing Character Strings**

For the comparison of character strings, each character position is compared in ASCII order. For example, \(a\) is less than \(b\).

If character strings of differing length are compared, the shorter character string will be padded with blanks to have the same length as the other string and then the strings are compared.

**Logical Operators**

Logical operators compare two true/false values, and the result is another true/false value.

A *true* value is any number not equal to zero.
A *false* value is zero.
The three logical operators are *and*, *or*, *not*.

**And Operator** The *and* logical operator has a value of 1 if the value on the right and on the left of the *and* are both true, otherwise the *and* operator has a value of zero.

**Or Operator** The *or* logical operator has a value of 1 if either the left value or the right value or both values are true, otherwise the *or* operator has a value of zero.

**Not Operator** The *not* logical operator has a value of 1 if the value to the right of the *not* is false, otherwise the *not* operator has a value of zero.

**Logical Operator Order**

The *not* operator is a unary operator and is evaluated before all other operators except negation. Negation and logical *not* are evaluated left to right.

The *or* and *and* logical operators are evaluated after relational operators. The *and* is evaluated first and then the *or*.

If you are unsure of how a logical expression will be evaluated, use parenthesis to force your required evaluation ordering. It is a good practice to use parentheses whenever an expression contains more than one logical operator.

**Logical Operator Example**

\[ a + 2 = b + 3 \text{ and } d < 4 \]

is evaluated as:

- 1. \( a + 2 \) is compared to \( b + 3 \) resulting in *true* or *false* (1 or 0).
- 2. \( d \) is compared to \( 4 \) resulting in *true* or *false* (1 or 0).
- 3. If the result from step 1 and step 2 are both *true*, then the entire expression is *true* (giving a value of 1), otherwise the expression is *false* (giving a value of 0).
Parentheses
Parentheses can be used to control and alter the usual evaluation of numeric and character equations.

Operations inside parentheses are done first. Parentheses can be nested inside each other. The innermost parentheses are evaluated first. There is no limit to the number of nested parentheses.

Whenever an expression contains more than one relational or logical operator, you are encouraged to use parentheses to guarantee the expression will be evaluated in the order you intend. Parentheses do not add any time to the processing and can be quite helpful in understanding complicated expressions.

For example:

\[
(2 + 3) * (4 + 5) \quad \text{has a value of } 45 \\
2 + 3 * 4 + 5 \quad \text{has a value of } 19 \\
2 + (3 * 4) + 5 \quad \text{also has a value of } 19
\]
Constants

**DBMS/COPY** allows three types of constants:

- Character Strings
- Numeric Values
- Date Values

**Character Strings**

Character string constants consist of 1 to 200 characters. The character string constant is enclosed in single or double quotes. The following are examples:

```
'this is a constant'
"so is this"
```

To put a single quote inside a character string, put two single quotes in a row (" "). (Not a regular double quote mark “ ”) For example:

```
'this quoted ''string'' has embedded quotes'
```

**Numeric Values**

Numeric constants can be positive or negative numbers. For positive numbers the leading sign is optional. Numbers can be entered with or without a trailing decimal point and can contain up to 16 digits.

Numbers can also be written in exponential notation. The exponent, the power of ten by which the number to the left of the e should be multiplied, can be any number between -300 and 300. The e in the exponential form can be entered in upper or lower case (E or e).

Numeric constants can have any value between –9e+99 to 9e99 and are precise to 16 digits.

If you need to indicate that a value is missing use a period. Please see the pages on missing values for more information.

**Numeric Examples**

```
123.
123
16.7894067
-3.45123e-4
16.7894067e12
```
**Dates**

**DBMS/COPY** processes dates from January 1, 1000 to December 31, 5000. Dates are converted into the number of minutes before or after midnight January 1, 1900 and stored as a number. Dates before 1900 are stored as a negative number and dates after 1900 are stored as a positive number.

Constant dates are entered as character strings with enclosing single quotes and then followed by a lower case *d*.

Dates are usually entered as *ddmmmyyyy*, but they can also have an hour field (*ddmmmyyyy:hh*) and a minute field (*ddmmmyyyy:hh.mm*).

- **Dd** the day of the month
- **Mmm** the 3 character abbreviation for the month. The abbreviation can be entered in upper or lower case. The abbreviations are:
  - Jan January
  - Feb February
  - Mar March
  - Apr April
  - May May
  - Jun June
  - Jul July
  - Aug August
  - Sep September
  - Oct October
  - Nov November
  - Dec December
- **Yyyy** is the number of years. If the year is entered in two digits, the current century is added. The years from 1000 to 5000 are valid.
- **Hh** the number of hours. It is entered on a twenty-four clock.
- **Mm** the number of minutes. The number can be from 0 to 59.

Examples of valid dates entered as constants:

- `'12feb83'd`
- `'12Feb1983:14.34'd`
- `'1JAN2020:1'd`
Date constants can also be of the \textit{m/d/y:h.m} format.

Date constants in the \textit{m/d/y:h.m} format are entered as character strings with enclosing single quotes and then followed by an \textit{m}. For example,

\begin{verbatim}
'2/12/83'm
'2/12/1983:14.34'm
'1/1/2020:1'm
\end{verbatim}

\textbf{Variables}

Valid variables are variables already existing in the input database or variables created with Assignment Statements.

\textbf{DBMS/COPY} can process variable names of any length up to 150 characters. The names can contain any characters, including embedded blanks and special symbols. (There are no reserved words in DBMS/COPY.)

Variable names that begin with letters or underscores and which contain only letters, underscores or numbers can be entered directly into the program.

If the variable name contains any other characters or embedded blanks, or if the name begins with any character except a letter or underscore, the name must be enclosed in carets (\textasciicircum). The caret marks inform DBMS/COPY that the surrounded information represents the name of one variable. A variable name that is enclosed in carets can be used anywhere any other variable name can be used.

DBMS/COPY does not differentiate between upper and lower case letters. To DBMS/COPY, Name, NAME, and name all represent the same variable name. DBMS/COPY does remember the way the name was typed. When the variable name is written to the output database, it will have the same capitalization as was typed, if possible in the specified output system.

If you specify a variable name that is not allowed under the output system, DBMS/COPY will modify the name as necessary to make it acceptable. Each supported package describes what are valid variable names. The name changes will be reported in the log file.
Some examples of valid variable names are:

```
name
ship_add
address1
a
A
^1A^  ^A B C D^  ^sales$^  
```

There are no reserved words in DBMS/COPY. However, if the words `in` or `out` are to be used as variable names in an Assignment Statement, they should be placed in caret marks to avoid confusion with the `in=` and `out=` components of the Database Specification Statement.

DBMS/COPY supports character, numeric and date variables.

Within the Assignment Statement, the first use of a variable determines its type. For example,

```
vara = 'abc';  vara becomes a character variable, length 3.
varb = 'abc  ';  varb becomes a character variable, length 5.
varc = 123*3;  varc becomes a numeric variable.
^Var $d^ = log(test);  Var $d becomes a numeric variable.
```

Within, **DBMS/COPY** the following assignments will create a variable of type "date". Remember, dates are stored as the number of minutes since January 1, 1900, so to add 30 days to a number, you must add 30 *1440.

```
vard1 = julian(1,1,1990);
vard2 = duedate - 30*1440;
vard3 = julian(12,15,1994) + 60 * 1440;
```

Character variables can have a length between 1 and 200 characters.

Both constants and calculated values can be assigned to variables.

You cannot use any variable before it is assigned a value either by an Assignment Statement or in an input database.
Special Variables

The DBMS/COPY system has four special variables that always exist and have predefined meanings. These variables are not written out to databases but you can assign these values to other variables.

_n_ The variable _n_ is a numeric variable containing the current input record number.

today_ The current date and time. This variable has a value equal to the number of minutes since January 1, 1900.

dateonly_ The current date. This variable has a value equal to the number of minutes to today at midnight since January 1, 1900.

pi_ \(\pi\)

Special variables cannot be sent directly to output databases. To output the value of a special variable to a database, it must be assigned to a variable. For example,

\[
\text{count} = \_n\;_;
\]

The count variable can be written to an output database.

Any variable containing the value of either the _today_ or the _dateonly_ special variable contains the numeric value discussed above. Within DBMS/COPY Plus for DOS, the variable can be set to a vartype of date, if desired, and then formatted in any of the date formats. Within DBMS/COPY for Windows, they are already date variables.

Missing Values

Sometimes a database will contain records where some variables do not have a value. These records are said to contain missing values. The way a software system treats a missing value can have a significant impact on any calculations involving the variable.

DBMS/COPY recognizes missing values and transfers them intact from the input system to the output system. The manual for DBMS/COPY contains specific information on how the different supported systems treat missing values.

The missing value for characters is a blank string.
DBMS/COPY uses a period (.) as the place holder for missing numeric values. The missing numeric value is less than the smallest allowable number. All numbers are greater than the missing value.

Numeric expressions that process a missing value will have as a result a missing value. For example,

\[
4 + . = . \\
4 + 0 = 4
\]

Numeric functions that are undefined also result in missing values. For example,

\[
3 / 0 = . \\
sqrt{-1} = .
\]

**Concatenation**

Concatenation is the operation of appending one character string to the end of another resulting in one longer string.

The concatenation of 'abc' and 'def' is 'abcdef'.

In the following examples, the symbol ( ) is used to represent a blank space. All trailing blanks in the first string are moved to the end of the combined strings. For example, the concatenation of 'abc⋅⋅⋅' and 'def' is 'abcdef⋅⋅⋅'.

Embedded blanks and leading blanks remain in position.

The concatenation symbol is || (2 vertical bars) with no space between the two bars. For example,

\[
'abc' || 'def' = 'abcdef'
\]

Any number of character expressions can be concatenated as long as the resulting character expression is within the system limit of 200 characters.

**Blank Separation In Concatenation**

If you wish to have a blank separating the two strings then use ||| (3 vertical bars). This is useful when concatenating first and last names.
To get more blanks between the two strings use more vertical bars. The number of blanks separating the two strings is equal to the number of vertical bars minus two. In the following example, 5 bars will result in 3 spaces:

\[
\text{'abc'} \ | | | | \text{'def'} = \text{'abc} \cdots \text{def'}
\]

A mailing list database has the person’s name separated into the last name variable and first name variable. A single variable with the last name and first name combined is needed. The new variable should look like — last name, firstname. By using the concatenation feature we can add a comma after the last name, and a blank after the comma.

```plaintext
compute;
    in=maillist.db out=newlist.db;
    name = lname || ',' ||| fname;
run;
```

**Reserved Words**

Some software systems have reserved words. These are words that have a special meaning within a system and their use is restricted to the place where they have that special meaning.

For example, in some systems the word *if* can only be used in the *if* statement and not as a variable.

**DBMS/COPY** has no reserved words.

Some words have special meaning at a particular location. For example, the word *label* can be a variable, and it can also be used to start the *label* statement. When a variable’s name is either *in* or *out*, enclose the variable name in carets (^in^, for example) to avoid confusion with the *in* and *out* components of the Database Specification Statement.
Functions

**DBMS/COPY** has numeric, character, and miscellaneous functions available to the user for computing new variables. Functions can be included in expressions and as part of Assignment Statements. Numeric functions return a numeric value; character functions return a character value. Miscellaneous functions do not properly fit in either category.

For more information, including the proper syntax, on any function, see the information on the particular function. The name, category, and a brief description of all the functions are listed below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>Absolute value</td>
<td>Mathematical</td>
</tr>
<tr>
<td>Acos</td>
<td>Arc cosine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>Arcos</td>
<td>Arc cosine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>Arsin</td>
<td>Arc sine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>asin</td>
<td>Arc sine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>atan</td>
<td>Arc tangent</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>atan2</td>
<td>Arc tangent; separate sine and cosine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>betainv</td>
<td>Inverse beta distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>bondytm</td>
<td>Bond yield to maturity</td>
<td>Financial</td>
</tr>
<tr>
<td>capital</td>
<td>Capitalize a string</td>
<td>Character</td>
</tr>
<tr>
<td>ceil</td>
<td>Next larger whole number</td>
<td>Mathematical</td>
</tr>
<tr>
<td>center</td>
<td>center a string</td>
<td>Character</td>
</tr>
<tr>
<td>chr</td>
<td>one character string having an ASCII value</td>
<td>Character</td>
</tr>
<tr>
<td>compound</td>
<td>Single payment</td>
<td>Financial</td>
</tr>
<tr>
<td>cos</td>
<td>Cosine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>cosh</td>
<td>Hyperbolic cosine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>day</td>
<td>Day of the month</td>
<td>Date</td>
</tr>
<tr>
<td>deg_to_rad</td>
<td>Converts degrees to radians</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>digamma</td>
<td>Derivative of the logarithm of the Gamma function</td>
<td>Mathematical</td>
</tr>
<tr>
<td>erf</td>
<td>Error function</td>
<td>Mathematical</td>
</tr>
<tr>
<td>erfc</td>
<td>Complement of the error function (1 - erf(x))</td>
<td>Mathematical</td>
</tr>
<tr>
<td>exp</td>
<td>Power of e</td>
<td>Mathematical</td>
</tr>
<tr>
<td>floor</td>
<td>Next smaller whole number</td>
<td>Mathematical</td>
</tr>
<tr>
<td>fulldate</td>
<td>Convert number of days to number of minutes</td>
<td>Date</td>
</tr>
<tr>
<td>fuzz</td>
<td>If the value of the expression is within 1e-12 of an integer, then the integer is returned.</td>
<td>Mathematical</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Category</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>gamma</td>
<td>Gamma function</td>
<td>Mathematical</td>
</tr>
<tr>
<td>hour</td>
<td>Hour of the day</td>
<td>Date</td>
</tr>
<tr>
<td>index</td>
<td>Find location of one string in another string</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>int</td>
<td>Truncate decimal part of a number</td>
<td>Mathematical</td>
</tr>
<tr>
<td>intdate</td>
<td>Convert number of minutes to number of days</td>
<td>Date</td>
</tr>
<tr>
<td>invcosh</td>
<td>Inverse hyperbolic cosine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>invsinh</td>
<td>Inverse hyperbolic sine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>invtanh</td>
<td>Inverse hyperbolic tangent</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>irr</td>
<td>Internal rate of return</td>
<td>Financial</td>
</tr>
<tr>
<td>julian</td>
<td>Convert month, day and year to internal date</td>
<td>Date</td>
</tr>
<tr>
<td>lagN</td>
<td>Lags a variable N records</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>left</td>
<td>Left justify string</td>
<td>Character</td>
</tr>
<tr>
<td>length</td>
<td>Length of the character string</td>
<td>Character</td>
</tr>
<tr>
<td>lfact</td>
<td>Natural logarithm of the factorial (ln(n!))</td>
<td>Mathematical</td>
</tr>
<tr>
<td>lgamma</td>
<td>Natural logarithm of the Gamma function</td>
<td>Mathematical</td>
</tr>
<tr>
<td>log</td>
<td>Natural log</td>
<td>Mathematical</td>
</tr>
<tr>
<td>log10</td>
<td>Base 10 log</td>
<td>Mathematical</td>
</tr>
<tr>
<td>lower</td>
<td>Convert string to all lower case</td>
<td>Character</td>
</tr>
<tr>
<td>max</td>
<td>Largest value of a list</td>
<td>Mathematical</td>
</tr>
<tr>
<td>mdy</td>
<td>Convert month, day and year to internal date value</td>
<td>Date</td>
</tr>
<tr>
<td>mean</td>
<td>Average of a list</td>
<td>Mathematical</td>
</tr>
<tr>
<td>min</td>
<td>Minimum value of a list</td>
<td>Mathematical</td>
</tr>
<tr>
<td>minute</td>
<td>Minutes of the date</td>
<td>Date</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus</td>
<td>Mathematical</td>
</tr>
<tr>
<td>month</td>
<td>Month of the date</td>
<td>Date</td>
</tr>
<tr>
<td>mort</td>
<td>Series of payments</td>
<td>Financial</td>
</tr>
<tr>
<td>n</td>
<td>Number of non-missing values in a list</td>
<td>Mathematical</td>
</tr>
<tr>
<td>nmiss</td>
<td>Number of missing values in a list</td>
<td>Mathematical</td>
</tr>
<tr>
<td>npv</td>
<td>Net present value</td>
<td>Financial</td>
</tr>
<tr>
<td>poisson</td>
<td>Poisson probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>pow</td>
<td>Raises one number to the power of another</td>
<td>Mathematical</td>
</tr>
<tr>
<td>probbeta</td>
<td>Beta probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probbnml</td>
<td>Binomial probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probchi</td>
<td>Chi-square probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>prof</td>
<td>Probability of an F distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probgam</td>
<td>Gamma probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probhypr</td>
<td>Hypergeometric probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probinvb</td>
<td>Inverse binomial probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Category</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>probinvc</td>
<td>Inverse chi-square probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probinvf</td>
<td>Inverse F probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probinvn</td>
<td>Inverse normal probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probinvt</td>
<td>Inverse student’s t probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probit</td>
<td>Inverse normal probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probn</td>
<td>Probability of a normal distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probnegb</td>
<td>Negative binomial probability distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probnorm</td>
<td>Probability of a normal distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>probt</td>
<td>Probability of a student t distribution</td>
<td>Probability</td>
</tr>
<tr>
<td>put</td>
<td>Converts variable to string based on format</td>
<td>String</td>
</tr>
<tr>
<td>qtr</td>
<td>Quarter of the year</td>
<td>Date</td>
</tr>
<tr>
<td>quarter</td>
<td>Quarter of the year</td>
<td>Date</td>
</tr>
<tr>
<td>rad_to_deg</td>
<td>Converts radians to degrees</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>random</td>
<td>Uniformly distributed random numbers</td>
<td>Probability</td>
</tr>
<tr>
<td>rannor</td>
<td>Normalized random numbers</td>
<td>Probability</td>
</tr>
<tr>
<td>right</td>
<td>Right justify a string</td>
<td>Character</td>
</tr>
<tr>
<td>round</td>
<td>Rounds values to a specified level of precision</td>
<td>Mathematical</td>
</tr>
<tr>
<td>saving</td>
<td>Series of payments</td>
<td>Financial</td>
</tr>
<tr>
<td>search</td>
<td>Find location of one string in another string</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>sign</td>
<td>Sign of the number</td>
<td>Mathematical</td>
</tr>
<tr>
<td>sin</td>
<td>Sine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>sinh</td>
<td>Hyperbolic sine</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>sqrt</td>
<td>Positive square root</td>
<td>Mathematical</td>
</tr>
<tr>
<td>string</td>
<td>Convert number to character string</td>
<td>Character</td>
</tr>
<tr>
<td>stringpart</td>
<td>Retrieves the requested non-blank part of the string</td>
<td>Character</td>
</tr>
<tr>
<td>subcol</td>
<td>Retrieve partial string based on starting column and length</td>
<td>Character</td>
</tr>
<tr>
<td>substr</td>
<td>Retrieve partial string based on starting and ending column. Also used to store into a string</td>
<td>Character</td>
</tr>
<tr>
<td>sum</td>
<td>Total of a list of numbers</td>
<td>Mathematical</td>
</tr>
<tr>
<td>tan</td>
<td>Tangent</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>tanh</td>
<td>Hyperbolic tangent</td>
<td>Trigonometric</td>
</tr>
<tr>
<td>trigamma</td>
<td>Second derivative of the logarithm of the Gamma function</td>
<td>Mathematical</td>
</tr>
<tr>
<td>uniform</td>
<td>Uniformly distributed random numbers</td>
<td>Probability</td>
</tr>
<tr>
<td>upcase</td>
<td>Convert string to all upper case</td>
<td>Character</td>
</tr>
<tr>
<td>upper</td>
<td>Convert string to all upper case</td>
<td>Character</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>value</td>
<td>Numeric value for a character string</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>valuelabel</td>
<td>Retrieves the formatted values of a variable which has a value list</td>
<td>Character</td>
</tr>
<tr>
<td>weekday</td>
<td>Day of the week</td>
<td>Date</td>
</tr>
<tr>
<td>year</td>
<td>Year (4 digit year)</td>
<td>Date</td>
</tr>
<tr>
<td>Year2digit</td>
<td>Year (2 digit year)</td>
<td>Date</td>
</tr>
</tbody>
</table>

**Syntax**

\[
\text{function}(\text{parameters})
\]

**Syntax Elements**

- **function**
  - The function name.

- **(parameters)**
  - The argument or arguments the function uses.
  - Each function requires a certain type of expression as parameters. The parameters are always enclosed in parentheses.

**Numeric Functions**

Numeric functions are those functions which return a numeric value. They have been divided into five categories, Date, Financial, Mathematical, Probability, and Trigonometric. There is general information about each category of numeric function below, and then the numeric functions are listed in alphabetic order.

**Date Functions**

DBMS/COPY stores every date value as the number of minutes since (or before) January 1, 1900. In the date function definitions, the term *expression* means a DBMS/COPY date value unless otherwise defined in the function definition. The date value can be expressed either in one of the date formats found in the DBMS/COPY Manual or as the numeric expression of the number of minutes since January 1, 1900.

Within DBMS/COPY, date arithmetic results will result in an internal date value. You do not need to use the Vartype Statement to force the date variable type. In the date function examples, *indate* is a variable...
containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM).

The date functions are day, fulldate, hour, intdate, julian, mdy, minute, month, qtr, quarter, weekday, year, and year2digit.

**Financial Functions**

In the financial function definitions, the function parameters are numeric expression unless otherwise defined in the definition. Additional limits on some of the parameters are specified in the definitions. Some of the functions can be used to calculate more than one type of value. These functions are identified in the text. For these functions, the user supplies three of the four function parameters and gives one as a missing value. The function will calculate the missing value. The financial functions are bondytm, compound, irr, mort, npv, and saving.

**Mathematical Functions**

In the Mathematical function definitions, the term expression means any numeric expression unless otherwise defined in the function definition.

The upper limits on the arguments of some of the functions are due to the limitations in the size of numbers that a PC can calculate. For example, the largest number that many machines can calculate is 1.8 E 308. This is the approximate value of \( \exp(700) \), so the \( \exp \) function’s limit is given as approximately 700. Other computers may have different maximum numbers. Thus the upper limit on some of the function arguments is machine dependent.

Mathematical functions are abs, ceil, digamma, erf, erfc, exp, floor, fuzz, gamma, int, lfact, lgamma, log, log10, max, mean, min, mod, n, nmiss, pow, round, sign, sqrt, sum, and trigamma.

**Probability Functions**

The function arguments for the Probability function definitions are numeric expression unless otherwise defined in the definition. Additional limits on some of the arguments are specified in the definitions. Some of the arguments are required to be integers. If so, and a non-integer is used as the argument, it will be truncated to an integer.
The probability functions are \textit{betainv, poisson, probbeta, probnml, probchi, probf, probgam, probhypr, probinwb, probinvc, probinuf, probinun, probit, probinvt, probn, probnorm, probneg, probt, random, uniform,} and \textit{rannor}.

\textbf{Trigonometric Function}

In the trigonometric function definitions, the term \textit{expression} means any numeric expression unless otherwise stated in the function definition. All the angles used in these functions are measured in radians. (180 degrees equals \(\pi\) radians.) For your convenience in using the functions, two conversion functions have also been included to convert degrees to radians and radians to degrees.

Trigonometric functions are \textit{acos, arcos, arsin, asin, atan, atan2, cos, cosh, deg\_to\_rad, invcosh, invsinh, invtanh, rad\_to\_deg, sin, sinh, tan,} and \textit{tanh}. 
Numeric Function Listing

abs(expression)
Numeric Function Category: Mathematical
Returns the absolute value (positive value) of the expression. If
the expression is negative, the sign will be reversed.

abs(-100) will return 100

acos(expression) or arcos(expression)
Numeric Function Category: Trigonometric
The acos and arcos functions are identical. They return the
angle between -π/2 and π/2 whose cosine equals the expression.
The angle measure is expressed in radians.
Argument Limits: -1 <= expression <= 1

acos(.5) will return 1.0471976 (which is π/3 radians or 60°)

arsin(expression) or asin(expression)
Numeric Function Category: Trigonometric
The arsin and asin functions are identical. They return the arc
sine of the expression. The arc sine is the angle between 0 and π
whose sine equals the expression. The angle measure is
expressed in radians.
Argument Limits: -1 <= expression <= 1

arsin(.5) will return .5235988 (which is π/6 or 30°)

atan(expression)
Numeric Function Category: Trigonometric
Returns the arc tangent of the expression. The atan function
returns the angle between -π/2 and π/2 whose tangent equals the
expression. The angle measure is expressed in radians. The atan
function cannot be used when the tangent is undefined. Use this
function when you have calculated values for the tangent. Use
the atan2 function when you have values for the sine and cosine
of the angle.

atan(1) will return .785398 (which is π/4 or 45°)
atan2(expression1,expression2)
Numeric Function Category: Trigonometric
Returns the arc tangent (measure) of the angle between -π and π
with sine = expression1 and cosine = expression2. The angle
measure is expressed in radians. Use this function when you
have the sine and cosine values for the angle. Use atan when you
have a calculated value for the tangent of the angle. The
advantage to atan2 is that it will recognize and return the angle
even when the tangent is undefined. Also, it will use the signs of
the arguments to identify the quadrant of the returned angle.

atan2(1,0) will return 1.5707963 (which is π/2 or 90°)

betainv(p,α,β)
Numeric Function Category: Probability
Returns the quantile from a beta distribution with probability of p
and parameters α and β. Betainv is the inverse of the probbeta
function.

Argument Limits
0 <= p <= 1
α >~ 0
β >~ 0

betainv(.784,1,3) will return .4

bondytm(price,rate,maturity,evaluation date)
Numeric Function Category: Financial
Returns the yield to maturity of a bond. Bondytm assumes semi-
annual coupon payments. The function arguments are:

price: the price paid for the bond as a percent of
       face value
rate: the yearly coupon rate
maturity: the date that the bond will come due
evaluation date: the date from which the yield should
                 be calculated.

bondytm(80.625,.0875,'15sep2008'd,'20feb1985'd) returns .1108
**ceil(expression)**

Numeric Function Category: Mathematical

Returns the smallest integer equal to or larger than the expression.

*ceil(4)* will return 4
*ceil(4.1)* will return 5

**compound(amount,future,rate,periods)**

Numeric Function Category: Financial

The compound function computes values for a single payment earning compound interest. The user supplies any three of the four parameters and gives one as a missing value. DBMS/COPY calculates the missing value. The function parameters are

- **amount**: the amount of the single payment
- **future**: the amount including all interest earned, at the end of the term (the future value)
- **rate**: the interest rate earned during each compounding period. For example, if the annual interest rate is .06 compounded monthly, the rate is .06/12 or .005.
- **periods**: the number of compounding periods until the end of the term. For example, if the interest will be compounded monthly for three years, the number of periods is 36.

The formula used to calculate the compound function is

\[
future = amount \times (1 + rate)^{periods}.
\]

*compound(500,2000, , 12)* will return .12246
*compound(500, , .005, 36)* will return 598.34

**cos(expression)**

Numeric Function Category: Trigonometric

The cos function returns the cosine of the expression. The expression must be in radians.

*\(\cos(0.5236)\) (which is \(\cos(\pi/6)\) ) will return .8660 (which is \(\frac{\sqrt{3}}{2}\))
cosh(expression)
Numeric Function Category: Trigonometric
Returns the hyperbolic cosine of the expression.

cosh(1.0472) will return 1.6002

day(expression)
Numeric Function Category: Date
Returns the day of the month. The number will be between 1 and 31. If indate is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

day(indate) will return 12

deg_to_rad(expression)
Numeric Function Category: Trigonometric
Converts the expression in degrees to an equivalent number of radians.

deg_to_rad(90) will return 1.5707963 (which is π/2 )

digamma(expression)
Numeric Function Category: Mathematical
Returns the derivative of the natural logarithm of the Gamma function of the expression, with respect to the expression. The expression must be a number greater than 0.

digamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}

digamma(1) will return -0.57722  digamma(5) will return 1.50611

erf(expression)
Numeric Function Category: Mathematical
Returns the error function value for the expression.

The error function is \( \frac{2}{\sqrt{\pi}} \int_0^{\text{expression}} e^{-x^2} dx \)

erf(1) will return 0.842701
erf(-0.8) will return -0.74210
erfc(expression)
Numeric Function Category: Mathematical
Returns the complement of the error function.

\[ erfc(x) = 1 - erf(x); \]
\[ erfc(1) \text{ will return } .157299 \]

exp(expression)
Numeric Function Category: Mathematical
The \( \text{exp} \) function returns \( e \) (about 2.718281828) raised to the power of the expression. This is the inverse of the log function. The upper limit of the expression is approximately 700.

\[ \text{exp}(2) \text{ is equivalent to } e^2 \text{ and will return } 7.3890561 \]

floor(expression)
Numeric Function Category: Mathematical
The largest integer equal to or smaller than the expression.

\[ \text{floor}(45) \text{ will return } 45 \]
\[ \text{floor}(-1.5) \text{ will return } -2 \]

fulldate(expression)
Numeric Function Category: Date
This function is provided to assist in date arithmetic. If you want to add a number of days to a given date, you must first convert the number of days to the number of minutes in those days. \( \text{Fulldate} \) performs this conversion. The expression is a number representing the number of days. \( \text{Fulldate} \) multiplies the expression by 1440, the number of minutes in a day. This is the inverse of the \( \text{intdate} \) function.

\[ \text{fulldate}(30) \text{ will return } 43200 \]
**fuzz(expression)**

Numeric Function Category: Mathematical

Returns the integer value for expressions within 1\(^{-12}\) of the integer. Sometimes when performing a great many calculations slight errors can occur. The `fuzz` function will determine if a number is extremely close to an integer. If it is close to the integer, the integer is returned; otherwise the original number is returned.

- `fuzz(1.0002)` will return 1.0002
- `fuzz(1.00000000000001)` will return 1.0

**gamma(expression)**

Numeric Function Category: Mathematical

Returns the Gamma \(\Gamma\) function for the expression. The expression must be greater than 0. The upper limit on the expression is approximately 160. If \(a\) is an integer, \(\text{gamma(a)} = \Gamma(a) = (a-1)!\). For all positive values \(\alpha\),

\[
\Gamma(\alpha) = \int_{0}^{\infty} x^{\alpha-1} e^{-x} \, dx
\]

- `gamma(5)` will return 24

**hour(expression)**

Numeric Function Category: Date

If the hour function is given a date value that includes a time value, it will return the hour of the day. If `indate` is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

- `hour(indate)` will return 5

**int(expression)**

Numeric Function Category: Mathematical

The `int` function truncates all digits to the right of the decimal. This is true for both positive and negative expressions.

- `int(2.3)` will return 2
- `int(-2.3)` will return -2
**intdate(expression)**

Numeric Function Category: Date
This function is provided to assist in date arithmetic. To calculate the days between two dates, subtract one date from the other. The result will be the number of minutes between the dates. In order to find the number of days, divide the result by 1440, the number of minutes in a day. *Intdate* divides the expression by 1440. It is the inverse of the *fulldate* function.

intdate(43200) will return 30

**irr(periods,flow0, flow1, ... flowN)**

Numeric Function Category: Financial
Calculates the internal rate of return for the series of flows. Periods indicates the number of cash flows per interest period. If periods = 0, then continuous compounding is used. Periods must be >= 0. Cash flows can be positive or negative. Negative payment flows are outflows, positive payment flows are inflows. The *irr* function is related to the *npv* (net present value) function. When period flows are net for each period, one per period, if the series of cash flows is discounted at the rate given by the *irr* function, the net present value will be zero.

irr(1,-10000,2000,4000,7000,5000,3000) will return .2835

**invcosh(expression)**

Numeric Function Category: Trigonometric
Returns the inverse of the hyperbolic cosine for expression for values greater than or equal to 1.

invcosh(1.6002) will return 1.047

**invsinh(expression)**

Numeric Function Category: Trigonometric
Returns the inverse of the hyperbolic sine for expression.

invsinh(1.249) will return 1.047

**invtanh(expression)**

Numeric Function Category: Trigonometric
Returns the inverse of the hyperbolic tangent for values between -1 and 1.

invtanh(0.7807) will return 1.047
\textit{julian(expression1,expression2,expression3)}

Numeric Function Category: Date

The \textit{mdy} and \textit{julian} functions are identical. They return the internal software system numeric date value for a supplied month, day, and year. Expression1 is the month, expression2 is the day, and expression3 is the year. The year can be either two digits or complete four digits. If a two digit year is used, the system will assume it is in the current century. If the month, day or year are invalid, a missing value is returned.

\textit{julian(11,23,56)} or \textit{julian(11,23,1956)} will return the internal date value for November 23, 1956 at 00:00 hours which is 29924640.

Note: if \textit{julian(11,23,56)} is run on or after January 1, 2000 then this will be interpreted as November 23, 2056.

This function’s definition has changed slightly because of year 2000 issues. Please see the Year 2000 page for more information. Our suggestion is to use a 4 digit year.

\textit{lfact(expression)}

Numeric Function Category: Mathematical

Returns the natural logarithm of the factorial of the expression. The expression must be an integer greater than 0.

\textit{lfact(5)} will return 4.787492

\textit{lgamma(expression)}

Numeric Function Category: Mathematical

Returns the natural logarithm of the Gamma function of the expression. The expression must be greater than 0.

\textit{lgamma(5)} will return 3.178054

\textit{log(expression)}

Numeric Function Category: Mathematical

Returns the natural log (log to the base \(e\)) of the expression. This is the inverse of the \textit{exp} function. The expression must be greater than 0.

\textit{log(4)} will return 1.3862944
log10(expression)
Numeric Function Category: Mathematical
Returns the log to the base 10 of the expression. The expression
must be greater than 0.

log10(4) will return .60206

max(expression1, expression2,..., expressionN)
Numeric Function Category: Mathematical
Returns the largest value of the list of expressions. Expressions
that evaluate to missing values are ignored. There can be up to
240 parameters.

a = 3;
b = -4;
c = 5;
d = .;
max(a, b, c, d) will return 5

mdy(expression1,expression2,expression3)
Numeric Function Category: Date
The mdy and julian functions are identical. They return the
internal software system numeric date value for a supplied
month, day, and year. Expression1 is the month, expression2 is
the day, and expression3 is the year. The year can be either two
digits or complete four digits. If a two digit year is used, the
system will assume it is in the current century. If the month, day
or year are invalid, a missing value is returned.

mdy(11,23,56) or mdy(11,23,1956) will return the internal date
value for November 23, 1956 at 00:00 hours which is 29924640.

Note: if mdy(11,23,56) is run on or after January 1, 2000 then
this will be interpreted as November 23, 2056.

This function’s definition has changed slightly because of year
2000 issues. Please see the Year 2000 page for more information.
Our suggestion is to use a 4 digit year.
mean(expression1, expression2, ... , expressionN)
Numeric Function Category: Mathematical
The mean function returns the average of the expressions. The average is the sum of the expressions divided by the number of expressions. Missing values are skipped and not averaged. There may be up to 240 parameters.

\[ \text{mean}(1, , 3, 8) \text{ will return } 4 \]

min(expression1, expression2, ... , expressionN)
Numeric Function Category: Mathematical
Returns the smallest expression in the list. Missing values are skipped. There can be up to 240 parameters.

\[ \text{min}(3, -4, , 5) \text{ will return } -4 \]

minute(expression)
Numeric Function Category: Date
Given a software system date value with a time value, the minute function will return the number of minutes past the hour. If there is no minute part, zero will be returned. If indate is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

\[ \text{minute}(\text{indate}) \text{ will return } 23 \]

mod(expression1,expression2)
Numeric Function Category: Mathematical
The mod function divides expression1 by expression2 and returns the remainder. The mod function has the same results as the modulus operator (mod). The formula for the mod function is

\[ \text{mod}(x, y) = x - \left( \left\lfloor \frac{x}{y} \right\rfloor \times y \right) \]

\[ \text{mod}(5, 3) \text{ will return } 2 \]

month(expression)
Numeric Function Category: Date
Given a software system date value, month will return the number of the month. January is month 1. If indate is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

\[ \text{month}(\text{indate}) \text{ will return } 2 \]
mort(amount, payment, rate, periods)
   Numeric Function Category: Financial
   The mort function computes loan values for a series of payments. The user supplies any three of the four parameters and gives one as a missing value. The mort function calculates the missing value. The function arguments are:
   amount: the loan amount
   payment: the uniform amount paid per period
   rate: the interest rate per period (see the compound function for a more detailed explanation)
   periods: the number of payments.
   The formula used in the mort function is:
   \[ \text{periods} = \frac{\text{rate} \times \text{amount} \times (1 + \text{rate})^{\text{periods}}}{(1 + \text{rate})^{\text{periods}} - 1} \]

   \[ \text{mort}(5000, 800, . , 12) \text{ will return } .11806 \]
   \[ \text{mort}(5000, . , .1, 12) \text{ will return } 733.8166 \]

n(expression1, expression2, ... , expressionN)
   Numeric Function Category: Mathematical
   Returns the number of non-missing values in the expression list. There can be up to 240 parameters.

   \[ n(3, -4, . , 5) \text{ will return } 3 \]

nmiss(expression1, expression2, ... , expressionN)
   Numeric Function Category: Mathematical
   Returns the number of missing values in the expression list. There can be up to 240 parameters.

   \[ nmiss(3, -4, . , 5) \text{ will return } 1 \]

cmp(rate, period, flow1, flow2, ... , flowN)
   Numeric Function Category: Financial
   \(^\text{Npv}\) calculates the net present value for the specified cash flows discounted by the rate. Period indicates the number of cash flows for the given rate. If period = 0, then continuous compounding is used. Period must be >= 0. Cash flows can be positive or negative. Negative payment flows are outflows, positive payment flows are inflows.

   \[ \text{npv}(1,1,-500,200,300,200) \text{ will return } 80.015 \]
**poisson(λ,n)**

**Numeric Function Category: Probability**

Returns the probability that an observation from a discrete Poisson distribution function with a mean parameter λ (lambda) is less than or equal to n. The probability is calculated using the following summation \[ \sum_{x=0}^{n} \frac{e^{-\lambda} \lambda^x}{x!}. \]

**Argument Limits**

\[ \lambda \geq 0 \]

\[ n \geq 0, \text{n must be an integer} \]

\[ \text{poisson}(7,4) \text{ will return .172992} \]

**pow(expression1,expression2)**

**Numeric Function Category: Mathematical**

Returns the value of expression1 raised to the expression2 power.

\[ \text{pow}(2,3) \text{ 2}^3 \text{ and will return 8} \]
\[ \text{pow}(2.7, 4.2) \text{ 2.7}^{4.2} \text{ and will return 64.822802} \]

**probbeta(x,α,β)**

**Numeric Function Category: Probability**

Returns the probability from a continuous beta distribution with shape parameters α and β at value x. The density function for probbeta is \[ \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \]

**Argument Limits**

\[ 0 \leq x \leq 1 \]

\[ \alpha > 0 \]

\[ \beta > 0 \]

\[ \text{probbeta(.4,1,3) will return .784} \]
**probbnml**\((p, n, m)\)

Numeric Function Category: Probability

Returns the probability that an observation from a discrete binomial distribution function with probability parameter \(p\) and degree \(n\) is less than or equal to \(m\). The probability is calculated by

\[
\sum_{j=0}^{m} \binom{n}{j} p^j (1 - p)^{n-j}
\]

Argument Limits

\(0 \leq p \leq 1\)

\(n > 1\)

\(0 \leq m \leq n, \quad n \text{ and } m \text{ are integers}\)

**probbnml(.4,3,1)** will return .648

**probchi**\((x, df)\)

Numeric Function Category: Probability

Returns the probability of a random variable with a continuous \(\chi^2\) distribution function with \(df\) degrees of freedom falls below the \(x\) value. The probability is calculated with

\[
\frac{x^{(\nu/2)} e^{-\nu/2}}{2^{\nu/2} \Gamma(\nu/2)}
\]

degrees of freedom.

Argument Limits

\(df > 0\)

\(x > 0\)

**probchi(5,2) will return .91795**

**probf**\((expression, dfn, dfd)\)

Numeric Function Category: Probability

Returns the probability of a random variable with an \(F\) distribution. \(dfn\) represents the numerator degrees of freedom. \(dfd\) represents the denominator degrees of freedom.

Argument Limits

\(x > 0\)

\(dfn, dfd > 0, \text{ integers}\)

**probf(5,3,2) will return .8288**
probgam(x, η)
Numeric Function Category: Probability
Returns the probability of a random variable with a continuous gamma distribution with a shape parameter η falls below the x value. The probability density is \( e^{-x} \frac{x^{\eta-1}}{\Gamma(\eta)} \).

Argument Limits
\( \eta > 0 \)
\( x \geq 0 \)

probgam(7,5) will return .8270

probhypr(n, k, s, x)
Numeric Function Category: Probability
Returns the probability that a sample of size s from a discrete hypergeometric distribution of size n with k attributes will have x or fewer observations with the attribute.

Argument Limits
\( 1 < n \)
\( 1 \leq k \leq n \)
\( 1 \leq s \leq n \)
\( \max(0, k + s - n) \leq x \leq \min(k, s) \)

probhypr(6,3,4,1) will return .2

probinvb(x, n, m)
Numeric Function Category: Probability
Returns the inverse of the probmnl function when given the probability x and parameters n and m.

Argument Limits
\( 0 \leq x \leq 1 \)
\( n > 1 \)
\( 0 \leq m \leq n. \quad n \text{ and } m \text{ are integers} \)

probinv(.648,3,1) will return .4
probinvc \( p, df \)  
Numeric Function Category: Probability  
Returns the \( \chi^2 \) quantile value when given a probability value, \( p \), and degrees of freedom, \( df \). This function is the inverse of the \( probchi \) function when \( p = 1 - \text{probchi} \)  
Argument Limits  
\( 0 \leq p \leq 1 \)  
\( df > 0 \), integer  

\( \text{probinvc(.082085, 2) will return 5} \)

probinuf \( p, df_n, df_d \)  
Numeric Function Category: Probability  
Returns the F quantile value when given a probability value, \( p \) degrees of freedom of the numerator, \( df_n \), and the degrees of freedom of the denominator, \( df_d \). This function is the inverse of the \( probf \) function.  
Argument Limits  
\( 0 \leq p \leq 1 \)  
\( df_n, df_d > 0 \), integers  

\( \text{probinuf(.8288, 3,2) will return 5} \)

probinvn \( p \)  
Numeric Function Category: Probability  
The \( \text{probinv} \) and \( \text{probit} \) functions are identical. They return the normal quantile when given a probability value, \( p \), from a normal distribution \( (0,1) \). These functions are the inverse of the \( \text{probn} \) and \( \text{probnorm} \) functions. All resulting values will lie between 8 and -8.  
Argument Limits are: \( 0 \leq p \leq 1 \)  

\( \text{probinvn(.5792)} \) \( \text{will return .2} \)
probinv(p, df)
Numeric Function Category: Probability
Returns quantile given a probability value, \( p \), and degrees of freedom, \( df \), from a Student’s t distribution. This function is the inverse of the \( \text{probt} \) function.

Argument Limits
\[ 0 \leq p \leq 1 \]
\[ dfn \geq 0, \text{ integer} \]

\( \text{probinv}(0.7218, 2) \) will return .7

probit(p)
Numeric Function Category: Probability
The \( \text{probinv} \) and \( \text{probit} \) functions are identical. They return the normal quantile when given a probability value, \( p \), from a normal distribution \((0,1)\). These functions are the inverse of the \( \text{probn} \) and \( \text{probnorm} \) functions. All resulting values will lie between 8 and -8.

Argument Limits
\[ 0 \leq p \leq 1 \]

\( \text{probit}(0.5792) \) will return .2

probn(expression) or probnorm(expression)
Numeric Function Category: Probability
The \( \text{probnorm} \) and \( \text{probn} \) functions are identical. They return the probability that a random variable with a normal distribution \((0,1)\) falls below the expression value.

Argument Limits
\( expression \) can be any number

\( \text{probn}(0.2) \) will return .57926
**probnegb**(p, n, m)
Numeric Function Category: Probability
Returns the probability that an observation from a discrete negative binomial distribution with parameters $p$ and $n$ is less than or equal to $m$.

Argument Limits
$0 \leq p \leq 1$
$0 \leq m$
$0 < n$

`probneg(.56, 7, 4)` will return 0.156781

**probt**(expression, df)
Numeric Function Category: Probability
Returns the probability that a random variable with a Student's t distribution with $df$ degrees of freedom falls below the expression value.

Argument Limits
$df > 0$, integer
`expression` can be any number

`probt(.7, 2)` will return 0.7218

**qtr**(expression) or **quarter**(expression)
Numeric Function Category: Date
The `qtr` and `quarter` functions are identical. Given a software system date value, the functions will return the quarter of the year. For months 1, 2, 3 the quarter will be 1, and so on. If `indate` is a variable containing the date value represented by `'12feb1983:5.23'd` (February 12, 1983 at 5:23 AM),

`qtr(indate)` will return 1

**rad_to_deg**(expression)
Numeric Function Category: Trigonometric
Converts the expression in radians to an equivalent number of degrees.

`rad_to_deg(1.5707963)` will return 90
(1.5707963 radians = $\pi/2$ radians = 90)
**random(expression)**
Numeric Function Category: Probability
*Random* and *uniform* are identical functions that will return a uniformly distributed random number between 0 and 1. You can supply the starting seed for the random number generator or let the software system pick one based on the time clock. The starting seed is looked at only on the first use of the *uniform* function. To start with a seed we recommend at least a 5 digit number. To start a random sequence with the system clock use zero. The resulting random number can be multiplied by a constant and truncated or rounded. For example, $\text{floor}(1000 \times \text{random}(0))$ will yield a random number between 0 and 999, inclusive.

- **random(12345)** will begin with a supplied seed
- **random(0)** will start with random seed

**rannor(expression)**
Numeric Function Category: Probability
Returns a normally distributed random number with mean 0 and standard deviation of 1. The resulting random number may be positive or negative. You can supply the starting seed for the random number generator or let the software system pick one based on the time clock. The starting seed is looked at only on the first use of the *rannor* function. To start with a seed we recommend you use at least a 5 digit number. To start a random sequence with the system clock use zero.

- **rannor(12345)** will begin with a supplied seed
- **rannor(0)** will start with random seed

**round(expression1,expression2)**
Numeric Function Category: Mathematical
Returns the value of *expression1*, rounded to the precision specified by *expression2*. If *expression2* is omitted, the precision is 1.

Round is defined by $\text{floor}\left(\frac{\text{expression1}}{\text{expression2}} + .5\right) \times \text{expression2}$.

- **round(456.782,.1)** will return 456.8
- **round(456.782,.01)** will return 456.78
- **round(456.782,10)** will return 460
- **round(456.782)** will return 457
saving(future,payment,rate,periods)

Numeric Function Category: Financial
The saving function computes savings values for a series of uniform payments. The user supplies any three of the four parameters and gives one as a missing value. The saving function calculates the missing value. The function arguments are:
- future: the future value after all payments and interest
- payment: the uniform amount saved per period
- rate: the interest rate per period (see the compound function)
- periods: the number of interest periods and payments.

The formula used in the saving function is:

\[ future = \frac{payment \times ((1 + rate)^{periods} - 1)}{rate} \]

saving(5000,200, . , 12) will return .1258
saving(5000, . , .1, 12) will return 233.82

sign(expression)

Numeric Function Category: Mathematic
The sign function returns 1 if the expression is greater than zero, 0 if the expression is zero, and -1 if the expression is less than 0.

sign(-3) will return -1

sin(expression)

Numeric Function Category: Trigonometric
The sin function returns the sine of the expression. The expression must be in radians.

sin(.5236) (which is sin(π/6)) will return .5

sinh(expression)

Numeric Function Category: Trigonometric
Returns the hyperbolic sine of the expression.

sinh(1.047) will return 1.24
sqrt(expression)

   Numeric Function Category: Mathematical
   The sqrt function returns the square root of the expression. The expression must be non-negative. Note: the sqrt is faster and more exact than pow(expression,.5)

   sqrt(4) will return 2

sum(expression1,expression2, ..., expressionN)

   Numeric Function Category: Mathematical
   The sum function returns the total of the list of expressions. Missing values are skipped. The there can be up to 240 expressions.

   sum(3, -4, .., 5) will return 4

tan(expression)

   Numeric Function Category: Trigonometric
   The tan function returns the tangent of the expression. The expression must be in radians.

   tan(.78539) (which is tan(π/4)) will return 1

tanh(expression)

   Numeric Function Category: Trigonometric
   Returns the hyperbolic tangent of the expression.

   tanh(1.047) will return .7807

trigamma(expression)

   Numeric Function Category: Mathematic
   Returns the derivative of the digamma function of the expression. (This is the second derivative of the log of the gamma function.) The expression must be greater than 0.

   trigamma(2) will return .644934
uniform(expression)
Numeric Function Category: Probability
Random and uniform are identical functions that will return a uniformly distributed random number between 0 and 1. You can supply the starting seed for the random number generator or let the software system pick one based on the time clock. The starting seed is looked at only on the first use of the uniform function. To start with a seed we recommend at least a 5 digit number. To start a random sequence with the system clock use zero. The resulting random number can be multiplied by a constant and truncated or rounded. For example, floor(1000 * uniform(0)) will yield a random number between 0 and 999, inclusive.

uniform(12345) will begin with a supplied seed
uniform(0) will start with random seed

weekday(expression)
Numeric Function Category: Date
Given a software system date value, weekday will return the day of the week. Monday has the value of 1. If indate is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

weekday(indate) will return 6

year(expression)
Numeric Function Category: Date
Given a software system date value, the year function will return the number of years. If indate is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

year(indate) will return 1983

Note: Because of year 2000 issues this function’s return value has changed from previous versions of DBMS/COPY. It now returns a 4 digit year. Please see the year 2000 page for more information. For a 2 digit year, see the year2digit function.
**Year2digit(expression)**

Numeric Function Category: Date

Given a software system date value, the year function will return the two digit number of years. If `indate` is a variable containing the date value represented by '12feb1983:5.23'd (February 12, 1983 at 5:23 AM),

`year2digit(indate)` will return 83

Note: This function was added because of year 2000 issues. Please see the year 2000 page for more information. If you want a 4 digit year, see the `year` function.
Character Functions

In the explanations of the character functions, the word *expression* means any thing that evaluates to a character string unless otherwise defined. The character string expression must always be contained in quotes. In the examples that follow, the symbol ⋅ is used to represent blank spaces.

Some of the miscellaneous functions provide information about character strings, such as the length or the location of one string in another. See the chapter on the miscellaneous functions for more information.

*capital(expression)*

The *capital* function capitalizes the expression. All alphabetic characters are converted to lowercase except those preceeded by blanks and the first character. The function returns a string of the same length as the expression.

*lower('abc  DEF') will return Abc  Def *

*center(expression)*

The *center* function takes the leading and trailing blanks in a field and divides them evenly between leading and trailing. The function returns a string of the same length as the expression.

*center(' ⋅⋅AbCdEf123⋅⋅⋅⋅⋅⋅⋅⋅') will return ⋅⋅⋅⋅ AbCdEf123 ⋅⋅⋅⋅ ⋅⋅ *

*chr(number)*

The *chr* function is used to create a one character string with a specific ASCII value equal to the expression. Since control characters cannot be stored in a quoted character string, use the *chr* function to send a control character to a device (a printer) to execute a special command.

*chr(12) will result in a Ctrl L *

*left(expression)*

The *left* function takes the leading blanks in a field and puts them at the end of the string. The function returns a string of the same length as the expression.

*left(' ⋅⋅AbCdEf123⋅⋅⋅⋅') will return ‘AbCdEf123⋅⋅⋅⋅’*
lower(expression)
The *lower* function converts the expression to all lowercase. The function returns a string of the same length as the expression.

`lower('AbCdEf123')` will return `abcdef123`

put(expression, format)
The *put* function converts numeric, date, and character strings and formats into text strings and stores the text string into a variable. The text is available for processing. The formats that are available are listed in the information about the Format Statement in the Compute Module Reference section.

```plaintext
a = 123.34;
b = put(a,aster12.2);
The variable b will contain ******123.34
```

right(expression)
The *right* function takes the trailing blanks in a field and places them in the front. The function returns a string of the same length as the expression.

`right('···············AbCdEf123··············')` will return `··············123.34··············`.

string(expression)
The *string* function converts a numeric expression into a left justified character string of length 12. The number is converted using the best12. format. Missing values will return as a decimal point.

`string(-12.0)` will return `-12          `.

stringpart(characterstring,expression1)
The *stringpart* function is used to extract the Nth non-blank part of the string. Expression1 specifies the section number.

`stringpart('a   bcd    e',1)` will return `a`
`stringpart('a   bcd    e',2)` will return `bcd`
`stringpart('a   bcd    e',3)` will return `e`
subcol(characterstring, expression1, expression2)

The `subcol` function is used to extract characters from another character string. Expression1 specifies the beginning column, and expression2 specifies the length. If the length (expression2) is not entered, the substring will be from the selected column (expression1) to the last column in the string. Note: This function is slightly different than the `substr` function.

`subcol('abcde', 2, 3)` will return bcd
`subcol('abcde', 2)` will return bcde

substr(characterstring, expression1, expression2)

Expression1 and expression2 are integers representing columns in a character string. Expression1 specifies the selected starting column, and expression2 specifies the ending column. The `substr` function serves two purposes. The first purpose is to extract a string from a range of characters in a character expression. If the ending column (expression2) is not entered, the substring will be from the beginning column (expression1) to the last column in the string.

`substr('abcde', 2, 3)` will return bc
`substr('abcde', 2)` will return bcde

The second purpose is to store a character string into a range of positions in another character string, replacing the characters already there. To store into a character string, place the `substr` function on the left-hand side of the equal sign as an Assignment Statement.

```plaintext
a = 'abcdef';
substr(a, 3, 4) = '12';
the variable a will contain ab12ef.
```

upcase(expression) or upper(expression)

The `upper` and `upcase` functions are identical. They convert the expression to all uppercase. The function returns a string of the same length as the expression.

`upper('AbCdEf123')` will return ABCDEF123
value label (variable)

The `value label` function retrieves the formatted values of a variable which has a value list.

Some software packages (SPSS and Stata for example) allow you to maintain a list of display labels which are associated with the variable’s underlying values. The `value label` function allows you to retrieve the display value. The result of the `value label` function is a character string. The length is the length of the longest label.

For example, the variable `gender` is numeric and has the following two value labels:

1  Female
2  Male.

```
gender_char = value label (gender);
```

The variable `gender_char` will be character and have a length of 6. For records where gender has a 1, `gender_char` will have Female. For records where gender has a 2, `gender_char` will have Male.
Miscellaneous Functions

The functions in this category cannot properly be called either numeric or character functions. Some of them return numeric values that provide information about character strings. The `lagN` function can return either a numeric or character value.

### index(string1, string2)

The `index` and `search` functions are identical. They look in string1 for string2. The functions return the column of the first character of the match. If string2 is not in string1, the functions return 0. The `index` and `search` functions are case sensitive (upper and lower case are different). String1 and string2 are character strings.

```plaintext
index('2 inch wiggle widget', 'widget') will return 15
index('2 inch wiggle widget', 'hammer') will return 0
```

### lagN(variable)

The lagging function creates a stack of values for the specified variable which can be either character or numeric. The number of values on the stack is determined by N. N must be an integer. The stack is initialized to all missing values. When called, the function returns the top value on the stack and puts the value for the current record on the bottom of the stack. The `lagN` function is useful for storing and retrieving old values.

For example, if a database with 10 records has a variable called `x` containing the numbers 1 through 10, a program paragraph containing the statement `y = lag3(x);` would result in the following:

```
x    y
1    
2    
3    
4    
5    
6    
7    
8    
9    
10   
```
length(character string)

Returns the length of the character string excluding trailing blanks. The value returned is the location of the last non-blank character. Embedded blanks will be counted.

`length('a bc    ')` will return 4

search(string1,string2)

The `index` and `search` functions are identical. They look in string1 for string2. The functions return the column of the first character of the match. If string2 is not in string1, the functions return 0. The `index` and `search` functions are case sensitive (upper and lower case are different). String1 and string2 are character strings.

`search('2 inch wiggle widget','widget')` will return 15
`search('2 inch wiggle widget','hammer')` will return 0

value(expression)

The `value` function takes a character string composed of numerals and converts it to a numeric value. If the string is not a valid number, `value` will return a numeric missing value. The function will recognize as numbers strings that include correctly used dollar signs, commas, decimal points, and exponential notation.

```
a = '$2,123.45   ';  
b = value(a);  
b will have the numeric value of 2123.45  
c = 'one';  
d = value(c);  
d will have a missing value.
```
DBMS/Explorer

DBMS/Explorer is an interactive, easy to use tool for the exploration and mining of data. Whether you are viewing the data for the first time or the long term owner of the data, DBMS/Explorer lets you “play” with the data to gain insight and knowledge.

The tables and graphs can be dynamically drilled down into, enabling you to gain insight into specific data subsets.

DBMS/Explorer has unsurpassed power in value mapping — taking raw data values and combining them into useful and more understandable groups and names. Want to do a demographic study of different age brackets but, have too many raw ages? DBMS/Explorer can interactively create all the age groups you need.

We know that you can make tables and graphs in a multitude of other packages but the specialized data exploration capabilities within DBMS/Explorer are unmatched. DBMS/Explorer can export the results to the printer, internet and Excel so you can share your results.

DBMS/COPY Version 7 for Windows comes with a free license to DBMS/Explorer. These pages will give you a brief overview of DBMS/Explorer. The complete DBMS/Explorer manual is included online.
Let's Get Started

Enough of that, let's make a table and a graph. These examples use the `saledbf.dbf` dBase database found in the `files` subdirectory of the main DBMS/Explorer and DBMS/COPY directory. When you first start DBMS/Explorer, you will see the following dialog box. From here you can easily make a table, graph and execute a previously saved exploration.

The simplest way to make a table or graph is from the above dialog box. You can also use the Quick Table and Graph menu items in the File menu. Go ahead and click on the **Quickly Make Table** button. The following dialog box will appear.

Once you select the input file, DBMS/Explorer will scan the file to determine which variables are suitable for use as row, column and page by variables. The numeric variables will be available for statistics. If some variables must be excluded you will be informed.
After you select the file, DBMS/Explorer will scan the database and building a multiple dimension data cube. This cube can then be analyzed very quickly and interactively. The cube can also be saved to disk so the next time you won’t have to start from the raw database.

If you need more control over the variables which will be available for the rows and columns or which variables should be available for statistics, please see the section on “Reading Data”.

**Defining the Rows and Columns**

The following dialog box is used to specify which variables go on the rows, columns, and page by variables. It is automatically displayed after the database is scanned. It is the same dialog box as from the “Define Table” menu item in the “Table” menu.

DBMS/Explorer can display tables with any number of fields on the rows and columns. Fields can be nested inside other fields to any depth. The same field can be used more than once within a table.

To add a field, highlight it and click the appropriate Append or Nest button. Nested fields will be indented, approximating the look of the table. If you want to insert a field, highlight the existing field at the insert point and add the new one.

The Swap button will switch the row and column fields.

For this example, let’s put the “Customers” on the row and the “Product” on the column. To do this, highlight *Customer* and click **Append Row**. Highlight *Product* and click **Append Column**. The dialog box should now look like:
Defining the Cell Statistics

Now that the row and column fields have been defined, you get to specify the cell statistics and cell variables. The following dialog box will be displayed:

This is the central dialog box for specifying what gets calculated for each row/column field combination. This dialog box is described in detail in the “Cell Contents” description later in this manual. For right now let’s create a table with the sum of the amount fields. So click on the Select Statistical Variables button, which will bring up the following dialog box:
Click on the **AMT** variable name. This will cause a 1 to appear in the **Order** column. (If you want more variables, just click on them in the order desired.) We only want the one variable so click **OK** when done. This will take you back to the previous dialog box which will look like this:

We really don’t need to know record counts and instead would like to know just the sum. So click on the **Count** label to remove that statistic and then click on the **Sum** label to add that statistic. While we are at it, let’s put a 0 (zero) in the # of Decimals column since we don’t need to see the cents, check the **Use Commas** so we can see the thousands, and lastly type a $ (dollar sign) in the Text Left of Number column so we will know the results are dollars.
That's enough for now so click Ok. Now your first table will be displayed as follows.

On the following page, we will turn the table into a graph.
**Make it a Graph**

Now let's turn that into a stacked bar chart. Under the **Table** menu you will see a set of **Graphs** items, select the **Bar Chart – Stacked** item. You will then see this dialog box:

![Bar Chart Graph Setup](image)

We actually want to subdivide the bars by the **Product** and make the basic bar the **Customer** so change the two selections. Everything else is correct. Once you make the changes and click **Ok**, the following graph will appear.

![Stacked Bar Chart](image)
**Do a Drill Down**

Let’s say that given the previous graph, we want to further analyze the books. Who sold the books? This is best done with a drill down. The first step is to select the parts of the graph (or table) which we want to further analyze. So click on the legend item *books*. You will see a bunch of D letters appear on the graph, this shows us which cells were selected.

Under the Drill-View menu, select the Create Table by Drilling Down item. (The right mouse popup menu also has this.) This will then take you to the Select Row/Column Variables dialog box which you have seen before. For now, select the Salesman variable and put it on the row. Click Ok. In the Cell Statistics dialog box, just click Ok, DBMS/Explorer carries the statistics over from the graph and you should see the following table:
This is a table of who sold books.

*Dynamic Drilling*

Now that you can drill, wouldn’t it be nice to create the same table on different selected cells, like different products or customers? Take a look at the “Dynamic Drill Down” section of this manual for this information.
Value Lists

DBMS/Explorer has a powerful facility to control and modify the display text for the variable values. In the last table we made, the Salesman had their abbreviations shown. We should change that to their first name. You can get to the format dialog box two ways. The not as easy way is to use the Format Rules menu item under the Table or Graph menus. It’s not as easy because you have to specify the variable name. An easier way is to use the right mouse button over the field of interest. In this case, move the mouse over the Salesman field or a value for that field. Select the Edit Value List item. Did you know about the Right mouse button stuff? Pretty nice. The following dialog box will appear when you do the right click.

This dialog box and all of the sub-dialog boxes which branch off of it are described in detail later in the manual. For now, let’s just change the names by clicking:

After we change the names and click Ok. Our table will be redrawn as,
The formatting options are quite powerful: you can collapse values, do autoranging, setup ranges, text substrings, lookup databases and more. Please take the time to read the section.

**Output Options**

The tables can be output to the printer, Html or Excel. Graphs can be output to the printer, Html or a file.

If you used a by group, the Html output will generate a table of contents page with hyperlinks to each by group. The Excel output will generate a multiple page spreadsheet.

The table description can be saved to disk and read back later.

The data cube can be saved to disk too.

All of these options are under the “File” menu.
Dynamic Drill Down

What is Drilling Down?
Drilling down is taking an existing table or graph, selecting specific cells, and creating a new table or graph based on the records within those cells.

For example, if you have a table of Customers versus Salesman (shown below), suppose you wanted to know what products were bought by Customers: Interstate, Palmer and Skyline.

You would click on those three customer names (holding down the control key adds the customers to the current selection). As you click on the names, you will see the cells for that customer are highlighted. Your table would now look like:
The Drill-Views menu items are used to create drill down views. If you want to create a new table based on those three customers, select the Create Table. If you want to create a new graph based on those three customers, select one of the graph types under the Create Graph sub-menu item.

Let’s create a new table. A table of products. As you create the new table, notice how the program uses the existing cell statistics as the starting point for the new table.

The previous table shows the sales made to only those three customers.

By tiling the windows, you can put the original and the drill down tables next to each other.
Dynamic Drilling

Dynamic Drilling is having the sub-tables and sub-graphs (those made by drilling down) automatically updated as you change the set of cells selected in the parent table or graph. In our example, the Customer/Salesman table is the parent table and the Product table is the sub-table.

Let’s try dynamic drilling. First, turn on dynamic drilling by selecting the Dynamic Drilling menu item in the Drill-View menu. (This is a "checked" menu item so there is no dialog box.)

Now, click on a different customer. The following table shows selecting Farrell. The product totals shown are only for Farrell.

<table>
<thead>
<tr>
<th>SALESMAN</th>
<th>Total</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sun</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bedford</td>
<td>AMT</td>
<td>$239</td>
<td>$229</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Empire</td>
<td>AMT</td>
<td>$1,407</td>
<td>$1,407</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Farrell</td>
<td>AMT</td>
<td>$3,430</td>
<td>$7,029</td>
<td>$4,304</td>
<td>$1,407</td>
<td>$108,904</td>
<td></td>
</tr>
<tr>
<td>Horwin</td>
<td>AMT</td>
<td>$2,414</td>
<td>$2,414</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interstate</td>
<td>AMT</td>
<td>$2,159</td>
<td>$2,591</td>
<td>$12,006</td>
<td>$1,243</td>
<td>$1,982</td>
<td>$20,720</td>
</tr>
<tr>
<td>Mayfield</td>
<td>AMT</td>
<td>$1,504</td>
<td>$1,504</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Palmer</td>
<td>AMT</td>
<td>$5,250</td>
<td>$9,802</td>
<td>$1,773</td>
<td>$1,024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skyline</td>
<td>AMT</td>
<td>$1,089</td>
<td>$1,089</td>
<td>$10,549</td>
<td>$5,520</td>
<td>$16,525</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>AMT</td>
<td>$6,655</td>
<td>$10,018</td>
<td>$90,012</td>
<td>$12,985</td>
<td>$13,150</td>
<td>$90,626</td>
</tr>
</tbody>
</table>
The selected group can be extended by using the control and/or shift keys to create different groups of cells.

Cells don’t have to be just the rows, you can also select the columns and ranges of specific cells. For example, we can select salesman “bt”, customer “Interstate” and salesman “ss”, customer “Skyline”.

Cells can also be excluded by clicking on highlighted cells. You might need to use the control and/or shift keys to get the exact set you want.
Having Multiple Drill Downs Open At Once

You can have up to 32 drill downs off of one main table open at the same time.

Each drill down will be numbered 1 to 32. If the drill down is off of the main table it will be called Drill.#. In the tables above, they are labeled Drill.1. If you made a second drill down off the main table it would be labeled Drill.2.

Having multiple drill downs open at the same time is great for comparisons. You could for example, create a drill down for one salesman, and then create another drill down for displaying how other selected salesmen were doing. Note: in this case you probably wouldn’t want to do Dynamic Drilldown but instead would want to Refresh the second drilldown with the refresh items on the Drill-View menu. Dynamic drilldown would automatically update both drilldowns instead of leaving one untouched.

Drilling the Drills

Of course, you can drill the drill downs. In the example above, you can select the “pearls” product to create a table or graph of the sales by date. The window heading will show Drill.1.2 which will mean that this is a subdrill off of Drill.1. Note: the drill numbers will vary but the nesting of the numbers tells you what is going on.
Drilling Graphs

Graphs can be drilled too. You can click on a bar segment, bar label or legend label. In the example below, we clicked on the Product “books” in the legend.
DBMS/Analyst
Overview

DBMS/COPY is a powerful package for transferring data between different software programs. Its batch processing enables you to automate the transferring process. Your free license to DBMS/Analyst enhances the batch processing capabilities of DBMS/COPY.

The complete DBMS/Analyst manual is on-line. The following few pages are meant to give you brief taste of DBMS/Analyst's power. Here are just a few of the things you can do in DBMS/Analyst:

- Create Multiple Output Databases
- Read Multiple Databases
- Merge Multiple Databases
- Maintain Multiple Databases
- Read and write complex ASCII character files
- Read and write supported database formats
- Produce Simple or Fancy Reports
- Randomly Access Individual Records In PRODAS Databases
- Perform Mathematical Calculations
- Perform Date Arithmetic
- Manipulate Character Strings
- Control Program Flow
- Manipulate Arrays
- Any operation the user defines using the language
- Execution time debugger
- Macro Language
- HTML output
- Equivalents to the SAS data step, sort, freq, means, summary, univariate, tabulate, transpose, rsq, reg, stepreg, format and print procedures. There is also an equivalent macro language.

DBMS/Analyst contains the following modules:

Crosstab cross tables with one or more variables. It also can process questionnaires.

Program the data manipulation module -- equivalent to the SAS data step.

Regress calculates all possible regressions, stepwise regression, linear regression, correlation, and collinearity.

Report printout generator
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>Sorts databases in a specified variable order.</td>
</tr>
<tr>
<td>Tabulate</td>
<td>Sophisticated cross table generator.</td>
</tr>
<tr>
<td>Transpose</td>
<td>Transposes the records and the variables. The records (rows) become variables and the variables (columns) become rows.</td>
</tr>
<tr>
<td>Unistat</td>
<td>Generates 30 different univariate statistics. These statistics range from basic (mean, minimum), to sophisticated (kurtosis, t statistics).</td>
</tr>
<tr>
<td>Value</td>
<td>Builds and maintains user defined formats</td>
</tr>
</tbody>
</table>

DBMS/Analyst enables you to use the sophisticated multiple database programming, control statements, array processing and all other commands in all modules. For example, DBMS/Analyst’s `crosstab` module can build cross tables by merging databases and computing new variables on the fly. To accomplish that in SAS, you would have to run the data step to merge the databases and compute the new variable, create a needless temporary dataset and then use that temporary dataset to make a table.
DBMS/Analyst Operation Overview

Database Operations
An important feature of DBMS/Analyst is database accessing. Databases of any format can be sequentially processed or merged together. In addition, our own PRODAS databases can be accessed randomly.

Any number of databases can be merged or concatenated with one statement. For example, you can merge customers with invoices and have the program module tell you whenever the current record is the first record of a customer, last record for a customer, if the customer doesn't have any invoices at all, or worse yet there isn't a customer for a group of invoices. You can take twelve files containing last year's data and concatenate them together to create one large database. The statements that are associated with sequential database processing include: set, merge, update, by, first, and last.

Printout Operations for the Program Module
Generating printouts is probably one of the most time consuming parts of programming. You have to create a page heading, keep track of the page number, keep track of the current line number and go to a new page when the current page is full, format numbers, check for field overflows, format dates, position data in the proper column, and convert coded fields into more readable text. The program module makes the job a lot easier. The program module lets you create a page heading subroutine, tracks the page number, tracks the line number and automatically calls the heading subroutine when the page is full. Formatting numbers and dates is trivial. Fields are automatically checked for overflow and dates can be printed in many different formats. Data can be positioned in any column, and by using a predefined format any value can automatically be printed in more readable text. The file and put statements that apply directly to printouts.

Printing in Other Modules
Most of the analysis modules generate printed output. DBMS/Analyst can write the results in either ascii text or HTML. The printer statement specifies the destination on a module by module basis. You can setup system wide defaults so you don't have to keep using a printer statement in every module.
Read Data From Many Places

ASCII Files for the Program Module
Input from ASCII disk files is controlled by the `infile` and `input` statements. The `infile` statement names the disk file. The `input` statement defines how to read each line on the input file by name, format, and column for each variable. Since these statements are under program control, you can read almost any ASCII file.

Sequential Input From Databases
Sequential input from databases is controlled by the Set, Merge, and Update Statements.

The `Set` Statement names the databases that should be read in. The action is to read in the records from the first database, then the second, third and so on. If the Set Statement is followed by a By Statement, the set will be interleaved. Records for the first group will be read from the first database, then the second database, and so on.

The `Merge` Statement names the databases that should be read in. The action is to match the records from each database for each group. The By Statement defines the groups. The Merge Statement enables you to merge customers with invoices for example.

The `Update` Statement names an old master database and a database of changes. The action is to match the records of the master database with the records on the changes database and make the changes to the master database if matched.

Random Access
DBMS/Analyst can retrieve a specific record from a sequential PRODAS database without having to start at record 1. The `match` function will perform a binary search on a sorted PRODAS database looking for an exact match. For example, looking up a patient on a database sorted by patient name. The `getrec` function will load in a record from a sequential PRODAS database given a record number.

DBMS/Analyst can retrieve a specific record from a multikeyed PRODAS database without having to start at record 1. The `bread` statement will look up a record based on the value of any one of the keys. For example, if a patient database was maintained in both patient name and patient social security number then either one can be used for the random
inputting. The **bnext** and **bprev** statements step forward and backward in key order through a database from the current key position.

### Many Processing Statements

DBMS/Analyst has many powerful processing statement. The **If / Then** Statement is used to decide which program statements are executed. The **Array** Statement allows the user to set up a multi-dimensioned mapping over variables. This is advantageous when the same calculations must be performed on many variables. The **Dimension** Statement allows the user to set up a multi-dimensioned memory space for storing, reading, and writing numeric or character values. The **Gosub** or **Link** Statements are used to jump to a subroutine. The **Return** Statement is used to mark the completion of the subroutine and to return back to the calling location. The **Goto** Statement is used to jump to another statement. The **For** loop and **Do** loop allow the user to set up statements that will be executed a specified number of times. They use an index variable which is incremented or decremented for each execution. The **While** loop, and **Do While** loop allow the user to set up statements that will be executed as long as the specified condition is true. **Assignment** statements allow you to create new variables and change the value of variables. When calculating the new value, the arithmetic, relational, and logical operators can be used. Also, the vast set of numeric and character functions can be used. Parenthesis can be used to modify the normal order of calculation.

### By Groups in the Program Module

A very important feature of DBMS/Analyst is its ability to process multilevel by groups. For example, a time billing database is coded by job and employee. You need to generate totals and a new page for each job and you need totals for employee within each job. By using the program module's by group processing this is an easy task. In the following sample program#3,

- The **by** statement defines the two level sort order.
- The **first.** operator is used to find out if the current record is the first record for a job and if it is then go to a new page, zero the job total and print the page headings.
- The **first.** operator is used to find out if the current record is the first record for an employee and if it is, set the total to zero.
- For every time record add the hours to the job total and employee total.
The last. operator is used to determine if the current record is the last record for an employee and if it is, print the name and total.

The last. operator is used to find out if the current record is the last record for a job and if it is print the total for the job.

By group processing can also be used to find records in one database that do not have corresponding records in another database, or duplicate records, or many other multi-database or single database manipulations.

**By Groups in all other Modules**

The by statement is used to perform the module procedure on subgroups of data. Although data for many groups may exist on one database, each group is processed as though it was on a separate database. For example, data for multiple cities may exist on one database but you might want one cross table per city. The by statement informs the crosstab module that it should treat each city as a separate group (as though each city was on a separate database). Since DBMS/Analyst allows the modules to also merge databases together please see the by statement’s documentation for a description of how to have that statement serve both purposes.

**Output Phase**

**Output Databases in All Modules except Program Module**

The database manipulation, reporting and statistical modules define how they output databases. Normally, output databases contain results. Please see the module’s documentation output database options.

**Writing ASCII data in the Program Module**

Writing ASCII disk files is controlled by the **File** and **Put** Statements. The File Statement defines where the data should go. The File Statement is also used to define a page heading subroutine, and the number of lines on a page. The Put Statement defines what should be printed, where and how it should be formatted.

**Creating New Sequential Databases in the Program Module**

The creating of a new sequential database is controlled by the Create and Output Statements. The Create Statement defines the name of the new output database. There are parameters to exclude or include specified
variables from the created database. The Output Statement writes out a copy of the variables to the end of the database. The Drop, Keep, and Rename Statements are used to control the excluding, including, and renaming variables on output databases.

**Updating PRODAS Databases in the Program Module**

The append function appends a new record at the end of an already existing sequential PRODAS database. The update function replaces a specified record in an already existing sequential PRODAS database.

The badd statement inserts a new record into a multikeyed database. The keys are updated to reflect the new record. The bupdate statement replaces the current record with the new record. If none of the key values are changed, the location of the record will not change. If keys have changed, the old key is deleted and the new one inserted. The bput statement combines the features of the badd and bupdate statements. Bput looks at the previous bread to see if the record is new or old.

**Sample Program #1:**

```plaintext
program;
create def;
set abc; /* input database */
perinch = weight / height; /* process */
if age > 20 then do;
  output; /* output record */
  file c:\temp\abc.lst;
  put age @20 sex @40 'Weight Per Inch'
     @60 perinch;
  /* printed on the terminal */
end;
run;
```

In the above sample program, the statements are executed as a group once for each record on the abc database. For example,

- the first record is read
- the perinch assignment statement is executed
- the if statement is checked
- If it is true (execute the block from do; to end;), the current value of the variables is written to the def database, the put statement is executed, writing the specified variables and literals to the terminal.
- When the run is hit, the system returns back to the top of the program, gets the next record and executes the statements again.
This continues until the last record on the database is read.

**Sample Program #2:**

crosstab;
merge invoices invlines;
by customer / inv_number;
  /* one table per customer */
table productcode;
run;

In the above sample program, one cross table is produced for each customer. The slash in the by statement indicates that the merge needs to be done for each customer and inv_number but that the grouping should only be to the customer level.

**Sample Program #3 :**

program;
set time;
/* input database */
by job employee;
file c:\temp\abc.lst;
/* define by groups (sorted order of database) */
if first.job then do;
  /* check for first job, if so print new page, and zero job total*/
  put _page_;
  job_tot = 0;
end;
if first.employee then
  /* check for first employee, if so zero employee total */
  emp_tot = 0;
emp_tot+hours;
/* sum hours for employee */
job_tot+hours;
/* sum hours for job */
if last.employee then
  /* check for last employee, if so print employee total */
  put @10 employee @40 emp_tot 12.2;
if last.job then
  /* check for last job, if so print job total */
  put job @20 'Job Total - ' job_tot 12.2;
run;

Sample program #3 is the implementation of the program described in the by group section.
A
Abs function, 184
Access, 21
Access - Microsoft Interactive, 45, 75
Acos function, 184
Arcos function, 184
Arsin function, 184
ASCII
  Data Dictionary, 36
  Interactive Fixed Format, 38
  Interactive Free Format, 42
Output
  Interactive, 72
  Reading Interactive, 36
Asin function, 184
Assignment Statement, 113, 115
Assistants
  Batch Editor, 88
Atan function, 184
Atan2 function, 185

B
Batch Editor, 85
  Assistants, 88
Batch Files, 82, 95
Batch Paragraph, 22
Batch Programs, 82
Batch Syntax, 82
Betainv function, 185
Bondytm function, 185

C
Capital function, 206
Ceil function, 186
Center function, 206
Changes from V6, 12
Character Functions
  Capital, 206
  Center, 206
  Chr, 206
  Left, 206
  Lower, 207
  Put, 207
  Right, 207
  String, 207
  Stringpart, 207
  Subcol, 208
  Substr, 208
  Upcase, 208
  Upper, 208
  Chr function, 206
  Command Line
    Interactive, 78
    Spreadsheets, 79
  Comments, 97
  Compound function, 186
  Compute Module, 83, 100
  Compute Module Examples, 22, 103, 125, 126, 129, 152, 163
  Computing New Variables
    Equation Builder, 58
    Concatenation, 165, 176
    Constants, 165, 171
      Character Strings, 171
      Dates, 172
      Numeric Values, 171
    Contents Module, 106
    Cos function, 186
    Cosh function, 187
    Createformatvars Statement, 113, 118
    Creating Batch Programs, 82, 85

D
Data Information, 66
Database Specification Statement, 113, 120
  Output Control Options, 124
Database Viewer, 54
Datainfo Module, 108
Date Function Summary, 181
Date Functions
  Day, 187
  Fulldate, 188
  Hour, 189
  Intdate, 190
  Julian, 191
  Mdy, 192, 193
  Month, 193
  Qtr, 200
Quarter, 200
Weekday, 204
Year, 204
Year2digit, 205
Date Values, 172
Day function, 187
DBMS/Analyst, 229
DBMS/COPY Plus, 82, 85, 94
Deg_to_rad function, 187
Delete Statement, 113
Digamma function, 187
Drop Statement, 113, 128
Dropping Variables
Interactive, 55

E

Equation Builder, 58, 64
Erf function, 187
Erf Function, 188
Error Messages, 98
Executable Statements, 104
Exp Function, 188
Expressions, 164
  Structure, 164, 165

F

Federal Express, 15
File Extensions, 16
Financial Function Summary, 182
Financial Functions
  Bondytm, 185
  Erf, 186, 190
  Mort, 194
  Npv, 194
  Saving, 202
Floor Function, 188
Format Statement, 113, 130
Fulldate Function, 188
Functions, 165, 178
  Abs, 184
  Acos, 184
  Arcos, 184
  Arsin, 184
  Asin, 184
  Atan, 184
  Atan2, 185
Betainv, 185
Bondytm, 185
Capital, 206
Ceil, 186
Center, 206
Chr, 206
Compound, 186
Cos, 186
Cosh, 187
Date Summary, 181
Day, 187
Deg_to_rad, 187
Digamma, 187
Erf, 187
Erfc, 188
Exp, 188
Financial Summary, 182
Floor, 188
Fulldate, 188
Fuzz, 189
Gamma, 189
Hour, 189
Index, 210
Int, 189
Intdate, 190
Invcosh, 190
Invsinh, 190
Invtanh, 190
Irr, 190
Julian, 191
LagN, 210
Left, 206
Length, 211
Lfact, 191
Lgamma, 191
Log, 191
Log10, 192
Lower, 207
Mathematical Summary, 182
Max, 192
Mdy, 192, 193
Mean, 193
Min, 193
Mod, 193
Month, 193
Mort, 194
N, 194
Nmiss, 194
Npv, 194
Numeric Summary, 181
Poisson, 195
Pow, 195
Probability Summary, 182
Probbeta, 195
Probnml, 196
Probc, 196
Probgam, 197
Probhpr, 197
Probinv, 197
Probinv, 198
Probinv, 198
Probinv, 198
Probinv, 199
Probit, 199
Probn, 199
Probnegb, 200
Probnorm, 199
Probit, 200
Put, 207
Qtr, 200
Quarter, 200
Rad_to_deg, 200
Random, 201
Rannor, 201
Right, 207
Round, 201
Saving, 202
Search, 211
Sign, 202
Sin, 202
Sinh, 202
Sqrt, 203
String, 207
Stringpart, 207
Subcol, 208
Substr, 208
Sum, 203
Tan, 203
Tanh, 203
Trigamma, 203
Trigonometric Summary, 183
Uniform, 204
Upcase, 208
Upper, 208
Value, 211
Weekday, 204
Year, 204
Year2digit, 205
Fuzz function, 189

G

Gamma function, 189
General Statements, 113
Assignment Statement, 113, 115
Createformatvars Statement, 113, 118
Database Specification Statement, 113, 120
Delete Statement, 113
Drop Statement, 113
Drop Statement, 128
Format Statement, 113, 130
If Statement, 113, 136
Inodbc Statement, 113, 139
Keep Statement, 113, 141
Label Statement, 113, 143
Modify Statement, 113, 144
Obs Statement, 113, 146
Order Statement, 114, 147
Outodbc Statement, 114, 149
Rename Statement, 114, 151
Retain Statement, 114, 153
Run Statement, 114, 156
Select Statement, 114, 157
Stop Statement, 114, 159
Values Statement, 114, 160
Vartype Statement, 114, 161

H

Hour function, 189

I

If Statement, 113, 136
Index function, 210
Inodbc Statement, 113, 139
Installation, 14
Int function, 189
Intdate function, 190
Interactive
ASCII Writing, 72
Data Information, 66
Database Viewer, 54
Dropping Variables, 55
Equation Builder, 58, 64
Keeping Variables, 55
Microsoft Access - Reading, 45
Microsoft Access - Writing, 75
ODBC - Reading, 47
ODBC - Writing, 76
Ordering Variables, 55
Power Panel, 53
Quick Start, 1
Rename Variables, 55
Sort, 65
Spreadsheets, 31
Transfer Verification, 77
Writing Multiple Member Files, 71
Interactive Interface, 26
Interactive Options, 69
InvCOSH function, 190
InvSINH function, 190
InvTANH function, 190
IRR function, 190

J
Julian function, 191

K
Keep Statement, 113, 141
Keeping Variables
Interactive, 55

L
Label Statement, 113, 143
LagN function, 210
Left function, 206
Length function, 211
LFACT function, 191
Lgamma function, 191
Limitations, 23
Log File, 80, 92
Empty, 69
Log function, 191
Log10 function, 192
Lower function, 207

M
Mathematical Function Summary, 182
Mathematical Functions
Abs, 184
Ceil, 186
Digamma, 187
Erf, 187
Erfc, 188
Exp, 188
Floor, 188
Fuzz, 189
Gamma, 189
Int, 189
LFACT, 191
Lgamma, 191
Log, 191
Log10, 192
Max, 192
Mean, 193
Min, 193
Mod, 193
N, 194
NMISS, 194
Pow, 195
Probbeta, 201
Sign, 202
Sqrt, 203
Sum, 203
Trigamma, 203
Max function, 192
MDY function, 192, 193
Mean function, 193
Microsoft Access, 21
Interactive, 45, 75
Min function, 193
Miscellaneous Functions, 210
Index, 210
LagN, 210
Length, 211
Search, 211
Value, 211
Missing Values, 19, 165, 175
Mod function, 193
Modify Statement, 113, 144
Month function, 193
MORT function, 194
Multiple Databases, 229
Multiple Member Files, 30
Interactive, 71

N
N function, 194
New for Version 7, 12
Nmiss function, 194
Npv function, 194
Numeric Function Summary, 181

O
Obs Statement, 113, 146
ODBC, 21, 139, 149
Interactive, 47
ODBC - Writing
Interactive, 76
Open Database Connectivity
Interactive, 47
Operators
Arithmetic, 164, 166, 167
Logical, 165, 166, 168
Relational, 164, 166, 167
Options Module, 83
Variable Retension, 153
Order Statement, 114, 147
Ordering Variables
Interactive, 55
Outodbc Statement, 114, 149
Output Control Options, 124

P
Paragraph
Batch, 82
DBMS/COPY Plus, 95
Parentheses, 165, 170
Poisson function, 195
Pow function, 195
Power Panel, 53
Data Information, 66
Database Viewer, 54
Equation Builder, 58, 64
Sort, 65
Variable Information, 55
Probability Function Summary, 182

Probability Functions
Betainv, 185
Poisson, 195
Probbeta, 195
Probchnml, 196
Probchi, 196
Probf, 196
Probhgam, 197
Probhypyr, 197
Probinvb, 197
Probinvc, 198
Probinvf, 198
Probinvc, 198
Probinvt, 199
Probit, 199
Probn, 199
Probnegb, 200
Probnorm, 199
Probt, 200
Random, 201
Rannor, 201
Uniform, 204
Probbeta function, 195
Probchnml function, 196
Probbeta function, 196
Probf function, 196
Probgam function, 197
Probhypyr function, 197
Probinvc function, 197
Probinvf function, 198
Probinvf function, 198
Probinvt function, 199
Probit function, 199
Problem Solving, 24
Probn function, 199
Probnegb function, 200
Probnorm function, 199
Probt function, 200
Programming Language, 229
Put function, 207

Q
Qtr function, 200
Quarter function, 200
Quick Start, 1
Variable Names, 19
Variable Type
   Interactive, 56
Variable Types, 17, 18
Variables, 165, 173
   Formats, 131
   Length, 144
   Shorthand, 97
   Special, 165, 175
   Types, 161
Vartype Statement, 114, 161

W
   Web Site, 6
   Weekday function, 204

Y
   Year 2000, 10, 69
   Year function, 204
   Year2digit function, 205