13. Command Reference

Introduction

Epi Info 7 is easy to operate in interactive mode, but complex or repeated operations require saving the steps as programs. Programs (similar to "scripts" in other software) can be used to set up menus, guide and limit the data entry process, restructure data, and do analyses.

In Form Designer and Classic Analysis, programming consists of interacting with a series of dialogs that produce the actual program statements. Experienced users may want to edit the statements or type them directly in the Program Editor. For this reason, the details of command syntax are provided and include a definition of each command and its operation because a single command (e.g., EXECUTE) may be found in Form Designer, Classic Analysis, and Visual Dashboard. Commands are in module based chapters with notation of differences that may exist between program implementation. Some commands are only available in one or two programs. Check Commands are saved in Form Designer and executed in Enter. Classic Analysis commands are generated, edited, executed, and may be saved with the Program Editor from Classic Analysis.

Functions and operators appear within commands and are used for common tasks (i.e., extracting a year from a date, combining two numeric values, calculating duration between two dates, or converting numbers to text and vice versa).

Check Code Commands

ASSIGN

Description

This command assigns the result of an arithmetic or string expression to a variable.

Syntax

```
ASSIGN <variable> = <expression>

ASSIGN <variable> = <defined DLLOBJECT>!<script function>({<parameters>})
```

- The <variable> represents a variable in a database or a defined variable created in a program.
- The <expression> represents any valid arithmetic or string expression.
- The <defined DLLOBJECT> represents any variable defined as a DLL object.
- The <script function> represents the name of a class or method inside the DLL or WSC that returns the desired value to be assigned.
- The <parameters> represent one or more optional function parameters to be passed into the DLL or WSC (do not include the {} or <> symbols in the code; parenthesis are required).

Comments

This command assigns the value of an expression to a variable. The variable may be a database variable in a form or Data Table, a user-defined variable created by the DEFINE command, or a system variable.

Examples

Example 1: The patient's age is calculated using the date of birth and the date the survey was last updated. The example assumes a form exists with the following fields: DOB (Date), SurveyDate (Date), and Age (Numeric). The code below would appear in the AFTER section of the second date field to be filled in by the user.

```
IF NOT BirthDate = (.) AND NOT SurveyDate = (.) THEN
   ASSIGN Age = YEARS(BirthDate, SurveyDate)
END-IF
```

Example 2: The code below will automatically set the checkbox field 'Minor' to true when the value of the field 'Age' is below 18. The example assumes a form exists that has the following fields: Minor (Checkbox) and Age (Numeric). The code below would appear in the AFTER section of the Age field.

```
IF Age < 18 THEN
   ASSIGN Minor = (+)
END
```

Example 3: A field is assigned the value of a mathematical expression that is used to calculate body mass index. The example assumes a form exists with the following fields: BMI (Numeric), Weight (Numeric), and Height (Numeric). The code below would appear in the AFTER section of the last field to be entered. Note that Weight and Height in this formula are being measured in pounds and inches, respectively.

```
ASSIGN BMI = (Weight / (Height * Height)) * 703
```

Example 4: A patient ID field is automatically generated using the patient's last name, gender, and middle initial. The example assumes a form exists with the following fields: ID (Text), LastName (Text), Sex (Text), and MI (Text). The code below would appear in the AFTER section of the last of the above fields to be entered.

```
ASSIGN ID = LastName & " - " & Sex & " (" & MI & ")"
```

AUTOSEARCH

Description

AUTOSEARCH causes Enter to search for records with values in the specified fields that match ones in the current record. If a match is found, it can be displayed, edited, or ignored, and the current record can continue to be entered.

Syntax

```
AUTOSEARCH [<field(s)>]
```

- The <field(s)> represents one or more fields to search.

Comments

The results are displayed as a spreadsheet. If you have more records than can be viewed in a single screen, a scroll bar appears to the right of the spreadsheet. Use the mouse to see the additional matched records.

To quickly navigate to one of the matched records returned, double-click the intended **row**. Alternatively, move the cursor to the desired row and press **Enter** or click **OK**. Navigating to a matched record will discard any data entered to the new record. All fields will show data from the selected record. The record number indicator at the lower left will show the record number of the selected record. To avoid selecting any of the matched records, press **Esc** or click **Cancel** to return to the current new record. Data entry will continue for the new record.

Fields displayed from a search are determined as follows:

- If a single field is the key field, it will be displayed with as many other fields as possible.
- Multiple key fields (if any) will be displayed before any others.

Example

The AUTOSEARCH command is used to find duplicate entries during data entry. In this example, duplicates are identified by a matching first and last name as in a name-based registry system. The example assumes a form exists with the following fields: FirstName (Text) and LastName (Text). The code below would appear in the AFTER section of the second field to be filled in by the user.

```
AUTOSEARCH FirstName LastName
```

Note: When searching on multiple fields, put the AUTOSEARCH command in Check Code for a field after all the key values have been entered. In the example above, both FirstName and LastName are key fields and LastName is the last of the key fields to be entered. The AUTOSEARCH command should appear in the Check Code for LastName.

BEEP

Description

This command causes the computer to generate a beep sound. It is often used to emphasize a customized message or warning dialog during data entry.

Syntax

```
BEEP
```

Comments

Command must be typed into the Program Editor since it is not available using the Command Tree.

Example

The computer emits a beep when invalid data is detected in the Age field. This code should appear in the AFTER section of the Age field.

```
IF Age > 5 THEN
   BEEP
   DIALOG "Do not include records for children over 5."
END-IF
```

CLEAR

Description

CLEAR sets the field named to the missing value, as if it had been left blank. The command is used to clear a previous entry when an error has been detected or a change occurs. More than one field may be specified. CLEAR is frequently followed by a GOTO command, which places the cursor in position for further entry after an error.

Note: More than one field can be used with the CLEAR command.

Syntax

```
CLEAR [<field(s)>]
```

- The <field> represents the name of a field on the form. If more than one field is specified, a space will separate them.

Comments

CLEAR will delete the value only for the current record. CLEAR cannot be used in grid tables. In Classic Analysis, you must clear a variable to use the ASSIGN command to assign it a value of null (.).

Examples

Example 1: The code below prevents invalid data from being saved to the current record by erasing it as soon as it is detected. The example assumes a form exists with the following field: Age (Numeric). The code below would appear in the AFTER section of the Age field.

```
IF Age >= 18 THEN
   BEEP
   DIALOG "Do not include records for adults."
   CLEAR Age
END-IF
```

Example 2: The code below prevents an invalid date from being saved to the current record by erasing it as soon as it is detected. The example assumes a form exists with the following fields: DOB (Date) and SurveyDate (Date). The code below would appear in the AFTER section of the SurveyDate field.

```
IF (DOB > SurveyDate) OR (SurveyDate > SYSTEMDATE) THEN
  CLEAR DOB SurveyDate
  DIALOG "Invalid date detected. Please try again."
END-IF
```

Example 3: The code below prevents an invalid date from being saved to the current record by erasing it as soon as it is detected. By using a GOTO command to move the cursor back to the SurveyDate field, you are forced to keep entering data until valid data are detected. The example assumes a form exists with the following fields: DOB (Date) and SurveyDate (Date). The code below would appear in the AFTER section of the SurveyDate field.

```
IF DOB > SurveyDate THEN
   CLEAR SurveyDate
   GOTO SurveyDate
   DIALOG "Survey date invalid. Please try again."
END-IF
```

COMMENTS (*)

Description

In Check Code and Classic Analysis, the combination of backslash and asterisk in the beginning of a line of code and an asterisk and backslash at the end, as shown in some code samples, indicates a comment. Commented lines are not executed. This allows you to enter user-defined comments to identify tasks, describe variable names or code blocks for documentation, and to assist with trouble-shooting or debugging.

Syntax

```
/* <text>

*/
```

- The <text> represents any alphanumeric text as a comment or a block of code slated to be ignored.

Comments

The /* character must be placed in the first column of a line to be recognized as comment mark. For comments longer than one line, the */ characters should be added at the end of the code. Use comments to disable commands.

Examples

Example 1: Comments are used to note the date of the code's generation date, purpose, and author.

```
/* Written by Jason D. Veloper, MPH - 06/30/2010

The block of code below uses the YEARS function
*/
ASSIGN AGE = YEARS(DOB, SYSTEMDATE)
```

Example 2: Comments are used to disable certain commands from executing.

```
/* The next few lines are incomplete and are commented for later

DEFINE PatientID Numeric - Note: need to determine ID should be a

Text type variable.

LIST - ToDo: need to add the variables to list

*/
```

DEFINE

Description

This command creates a new variable. In Check Code, all user defined variables are saved in the DEFINEDVARIABLES section.

Syntax

```
DEFINE <variable> {<scope>} {<type indicator>}
```

- <variable> represents the name of the variable to be created. The name of the newly defined variable cannot be a reserved word. For a list of reserved words, see the List of Reserved Words section.
- <scope> is optional and is the level of visibility and availability of the new variable. <scope> must be one of the reserved words: STANDARD, GLOBAL, or PERMANENT. If omitted, STANDARD is assumed and a type indicator cannot be used. For information on defining a variable as a DLL OBJECT, see the DEFINE DLLOBJECT command.
- <type indicator> is required and cannot be used if <scope> is omitted. <type indicator> is the data type of the new variable and must be one of the following reserved words: NUMERIC, TEXTINPUT, YN, or DATEFORMAT. If omitted, the variable type will be inferred based on the data type of the first value assigned to the variable. However, **omitting field type is not recommended**. Once field type is defined, the variable type cannot be changed. An error will occur if you attempt to assign data of a different type to the variable.

Comments

A custom variable defined in Epi Info 7 might not have a predefined data type if the <type indicator> is omitted when the variable is defined. If the variable does not have a predefined type and has not been used, the variable will accept a value in any of the four data types (Text, Number, Date, Yes/No [Boolean]) that is assigned to the variable the first time. Thereafter, the variable takes on the data type of the value assigned and it's data type cannot be changed. However, omitting field type is not recommended. An error will occur if you attempt to assign data of a different data type. Various functions can be used to manipulate data, changing data type of values to match the data type of the variable. Some of these functions include FORMAT, TXTTONUM, TXTTODATE, and NUMTODATE.

Variable Scope

- **STANDARD** variables retain their value only within the current record and are reset when you load a new record. Standard variables are used as temporary variables behaving like other fields in the database. In Classic Analysis, Standard variables lose their values and definitions with each READ statement.
- **GLOBAL** variables retain values across related forms and when the program opens a new form, but are removed when you close the Enter program. Global variables persist while the program is executed. Global variables are also used in Classic Analysis to store values between changes of data source.

- **PERMANENT** variables are stored in the EpiInfo.Config.xml file and retain any value assigned until the value is changed by another assignment or the variable is undefined. They are shared among Epi Info programs (i.e., Enter, Classic Analysis, etc.) and persist even if the computer is shut down. Permanent variables in Classic Analysis may not have values that depend directly or indirectly on table fields. A <prompt/description> created for a permanent variable will exist for one session, and must be re-established each time it is used.

Type Indicators

- **TEXTINPUT** - Variables of this data type can receive any alpha-numeric characters including symbols and the output of functions (e.g., FORMAT).
- **NUMERIC** - Variables of this data type can receive numbers and the output of functions (e.g., TXTTONUM).
- **DATEFORMAT** - Variables of this data type can receive date values including the output of functions (e.g., TXTTODATE and NUMTODATE).
- **YN** - Variables of this data type can receive the Boolean values of (+) for Yes and (-) for No. Until an assignment is made, YN type variable values are (.) or missing.

Examples

Example 1: A variable is defined without <scope>, <type indicator>, and <prompt/description>. Standard scope thus becomes the default scope. As no type is specified, the variable can be assigned number, date, text, or Boolean data. However, once assigned a value, the data type of the variable becomes the data type of the value that has been assigned and may not be changed.

```
DEFINE BodyMassIndex NUMERIC
```

Example 2: A variable with standard scope and of type YN is defined.

```
DEFINE DoYouSmoke YN
IF DateSmokingStarted = (+) THEN
   ASSIGN DoYouSmoke = (+)
ELSE
   ASSIGN DoYouSmoke = (.)
END-IF
```

Example 3: A variable with standard scope and in date format is defined.

```
DEFINE DateOfBirth DATEFORMAT
```

Example 4: A variable with permanent scope is defined, but with no <type indicator>. As described in Example 1, the variable data type will be set with the first assignment of data and cannot be changed thereafter.

```
DEFINE StateID PERMANENT
```

Example 5: A variable with global scope and of type text is defined.

```
DEFINE PatientID GLOBAL TEXTINPUT
```

DEFINE DLLOBJECT

Description

This command creates a variable that represents an ActiveX object and a class within an ActiveX DLL (dynamic link library) file, ActiveX EXE (executable file), or a Windows Scripting Component (WSC) file.

Syntax

ActiveX DLL File

```
DEFINE <variable> DLLOBJECT "<ActiveX name>.<class>"
```

Windows Scripting Component (WSC) File

```
DEFINE <variable> DLLOBJECT "<filename>"
```

- The <variable> represents the name of the variable to be created. <variable> cannot be a reserved word.
- The <ActiveX name> represents the internal name of the ActiveX object that contains the class object for the DLL or EXE file.
- The <class> represents an internal class name that is defined within the ActiveX component.
- The <filename> represents the name of the Windows Scripting Component (WSC) file where the script component resides.

Comments

The ActiveX name may not be the same as the actual name of the dll (dynamic link library) or executable (exe) file. An ActiveX object is given a name when it's developed. This name is required to create the object. The ActiveX object (executable or dll) must be registered before it can be used. Windows Scripting Component (WSC) objects can also be used.

Examples

Example 1: A variable is created that points to a class object within the Epiweek DLL file. This variable can subsequently be used to find the Epi Week for any given date.

```
DEFINE Week DLLOBJECT "EIEpiwk.Epiweek"
```

Example 2: A variable is created that points to a class object within the GetGlobalUniqueID.WSC file. You can use this variable to assign a field a unique ID.

```
DEFINE Global_ID DLLOBJECT "GetGlobalUniqueID.WSC"
```

AFTER and END-AFTER

Description

This command divides Check Commands to be executed after data entry. All commands in the AFTER and END-AFTER block are executed *after* entering data to the field, page, or form.

Syntax

```
AFTER
```

- AFTER and END-AFTER are only used in Check Code.

Comments

This command enables actions to occur after accessing a form, page, or field. The default time to execute commands associated with a variable is after entry.

AFTER must start in the first position of the line and end with END-AFTER.

Example

The following commands represent the Check Code for a variable called Demo1.

```
BEFORE
   DIALOG "This is Before entry"
END-BEFORE

AFTER
   DIALOG "This is After entry"
END-AFTER
```

BEFORE and END-BEFORE

Description

This command divides Check Commands to be executed before data entry from those executed after entry. All commands in the BEFORE and END-BEFORE block are executed *before* data entry to the field, page, or form.

Syntax

```
END-BEFORE
```

- ENDBEFORE is only used in Check Code. Command buttons will not take ENDBEFORE since all code inserted will be executed when you click the button.

Comments

This command enables actions to occur before accessing a form, page, or field. The default time to execute commands associated with a variable is before entry.

BEFORE must start in the first position of the line and end with END-BEFORE.

Example

The following commands represent the Check Code that raise a dialog before the cursor enters the field before data entry, and then after the cursor leaves the field following data entry.

```
BEFORE
   DIALOG ”This is Before entry”
END-BEFORE

AFTER
   DIALOG ”This is After entry”
END-AFTER
```

CLICK and END-CLICK

Description

This command block is for fields that have a click event such as Command Buttons, Checkboxes, Legal Values, and Comment Legal fields. Not all field types support the CLICK command. Check Commands in the Click block are executed immediately upon clicking the field or clicking an item in a drop-down list.

Syntax

```
CLICK
   //Add commands here

END-CLICK
```

Comments

This command block enables actions to occur immediately upon clicking a field that supports the CLICK event.

CLICK must start in the first position of the line and end with END-CLICK.

Example

The following commands raise a dialog when the field is clicked.

```
CLICK
   DIALOG ”The checkbox was just clicked.”
END-CLICK
```

EXECUTE

Description

Executes a Windows or DOS program - either one explicitly named in the command or one designated within the Windows registry as appropriate for a document with a named file extension. This provides a mechanism for bringing up the default word processor or browser on a computer without first knowing its name. The EXECUTE command accepts a series of paths, separated by semicolons:

```
EXECUTE c:\epi_info\myfile.exe;d:\myfile.exe
```

If the first is not found, the others are tried in succession. In Check Code, the EXECUTE command can be placed in any command block, but is often used with a button. A button does not have a Before Entry section.

Syntax

```
EXECUTE <filename>

EXECUTE "<filename> <command-line parameters>"

EXECUTE NOWAITFOREXIT <filename>

EXECUTE NOWAITFOREXIT '<filename>'

EXECUTE NOWAITFOREXIT "<filename>"

EXECUTE WAITFOREXIT <filename>

EXECUTE WAITFOREXIT '<filename>'

EXECUTE WAITFOREXIT "<filename>"
```

- The <filename> represents the path and program name for .exe (filename for registered Windows programs) and .com (filename for MS-DOS binary executable) files.
- The <command-line parameters> represent any additional command-line arguments that the program can accept.
- When Wait for Command to Execute (modal) is specified, the command should run and Enter should continue running. When Wait for Command to Execute is not specified (non-modal), Enter should wait until the executed program closes before continuing. When EXECUTE is run modally, permanent variables are written before the command is executed and reloaded after it is executed.

Comments

If the name of an executable program, (e.g., ENTER.EXE, MYBATCH.BAT, or MYWEB.HTM) is given, the program will be run in a separate window. The window closes when the program terminates.

If the name given is not a program, but a file with an extension (the three characters after the ".") registered by Windows for displaying the document, the correct program to display the file will be activated (i.e., WRITEUP.DOC might cause Microsoft Word© to run and load the file on one computer). Usually .TXT will run NOTEPAD.EXE© or WORDPAD.EXE©, and image files will appear in a browser or in a graphics program. An .HTM file will bring up the default browser.

You will not need to supply the location or even the name of the program that will be run. These details are stored in the Windows registry for common file extensions. The same concept applies to Internet addresses (URLs). Give a URL (Universal Resource Locator) that ends in .HTM or begins with HTTP:// "http://www.cdc.gov/epiinfo/" and Windows brings up the default browser, connects to the Internet (if possible), and goes directly to the site indicated.

Examples

Example 1: A text file on the C drive is opened. The operating system will select an application to open the file.

```
EXECUTE "C:\logfile.txt"
```

Example 2: An executable file is run.

```
EXECUTE "C:\Windows\Notepad.exe"
```

Example 3: Windows Internet Explorer is run and passed http://www.cdc.gov/epiinfo as a command-line parameter. Because WAITFOREXIT is specified, Enter will not allow you to continue entering data until the browser window is closed.

```
EXECUTE WAITFOREXIT "http://www.cdc.gov/epiinfo"
```

GOTO

Description

This command can be used alone or in an IF statement to transfer the cursor to a named variable field.

Syntax

```
GOTO <event>
```

- The <event> can be a +1, -1, page number, or a field name.

Event and Description

- +1 Automatically saves the current page if changes have been made, and goes to the next page.
- -1 Automatically saves the current page if changes have been made, and goes to the previous page.
- <page number> Automatically saves the current page if changes have been made, and goes to the page indicated by the number.
- <field name> Goes to the field indicated. Automatically saves the current page if changes have been made, if the field is on another page.

Comments

GOTO will be ignored if it is in the Before or After Record event. The GOTO command skips all the variables in between unavailable for data entry or Read Only.

Examples

Example 1: The form will skip directly from one field to another based on certain user input. The example assumes a form exists that has the following fields: DoYouSmoke (Yes/No), PacksPerDay (Numeric), and HeartDisease (Yes/No). The code below would appear in the AFTER section of the DoYouSmoke field.

```
IF DoYouSmoke = (+) THEN
   GOTO HeartDisease
ELSE
   GOTO PacksPerDay
END-IF
```

Example 2: You will be taken to the second page on the form based upon an answer provided to a question on lung disease. The example assumes a form exists that has two pages and the following fields: LungDisease (Yes/No). The code below would appear in the AFTER section of the LungDisease field.

```
IF LungDisease = (-) THEN
   GOTO  2
END-IF
```

Example 3: You will be taken to the page immediately following the one they are on currently. The example assumes a form exists that has two pages and the following fields: DOB (Date) on page 1. The code below would appear in the AFTER section of the DOB field.

```
IF NOT DOB = (.) THEN
   GOTO +1
END-IF
```

UNHIDE

Description

HIDE - This command hides a field from the form and prevents data entry.

UNHIDE - This command makes a field visible and returns it to the status before it was hidden.

Syntax

```
HIDE [<field(s)>]

UNHIDE [<field(s)>]
```

- The <field(s)> represents one or more valid field names.

Comments

If no field name is specified, the current field (the one to which the Check Code block pertains) is assumed. Text fields can be hidden or unhidden to position messages on the screen, and the display of alternate messages. Label/Title fields cannot be hidden.

Examples

Example 1: Questions relating to pregnancy will not be displayed if the patient is male. The ELSE section of the IF command allows the fields to be unhidden if you go back and change your answer in the Sex field. The example assumes a form exists with the following fields: Sex (Text), Pregnant (Yes/No), and ChildBirth (Yes/No). The code below would appear in the AFTER section of the Sex field.

```
IF Sex = "M" THEN
   HIDE Pregnant
   HIDE ChildBirth
ELSE
   UNHIDE Pregnant
   UNHIDE ChildBirth
END-IF
```

Example 2: The field that has the HIDE command will be hidden. The example assumes a form exists with two pages and the following fields: LungDisease (Yes/No). The code below would appear in the AFTER section of the LungDisease field.

```
IF LungDisease = (-) THEN
   HIDE
END-IF
```

Example 3: Questions relating to pregnancy will not be displayed if the patient is a male. The example assumes a form exists with the following fields: Sex (Text), Pregnant (Yes/No), Complications (Yes/No), and ChildBirth (Yes/No). The code below would appear in the AFTER section of the Sex field.

```
IF Sex = "M" THEN
   HIDE Pregnant ChildBirth Complications
END-IF
```

Note: When hiding a field, it is important that be unhidden (with an If…Then…Else…) if the value entered is changed.

IF THEN ELSE

Description

This command defines conditions and one or more consequences which occur when the conditions are met. An alternative consequence can be given after the ELSE statement to be realized if the first set of conditions is not true. The ELSE statement is optional.

Syntax

```
IF <expression> THEN
    [command(s)]
END-IF

IF <expression> THEN
    [command(s)]
ELSE
    [command(s)]
END-IF
```

- The <expression> represents a condition that determines whether or not subsequent commands will be run. If the condition evaluates to true, the commands inside of the IF block will run. If the condition evaluates to false, the commands inside of the ELSE block will run instead. If no ELSE exists and the condition is false, then no commands inside of the IF block are run.
- The (command[s]) represents at least one valid command.
- The ELSE statement is optional and will run any code contained inside of it when the <expression> evaluates to false.

Comments

The IF statement is executed immediately if it does not refer to a database variable, any characteristic or attribute that can be measured, or if any defined variables have been assigned literal values.

Examples

Example 1: If you select "Male" for the patient's sex then the fields named Pregnancy and ChildBirth are hidden. The example assumes a form exists with the following fields all on the same page: Sex (Text), Pregnancy (Yes/No), and ChildBirth (Yes/No). The code below would appear in the AFTER section of the Sex field.

```
IF (Sex = "Male") THEN
    HIDE Pregnancy Childbirth
END-IF
```

Example 2: If the date of birth supplied by the user occurs prior to January 1, 1900, the Enter module provides a warning beep and a warning dialog indicating invalid input. However, if the date of birth supplied is on or after January 1, 1900, the GOTO command is executed instead taking you to the following page. The example assumes a form exists with the following fields: DOB (Date). It also assumes a page exists following the page where DOB resides. The code below would appear in the AFTER section of the DOB field.

```
IF DOB < 01/01/1900 THEN
   BEEP
   DIALOG "Warning: Invalid date of birth detected"
ELSE
   GOTO +1
END-IF
```

Example 3: The date of birth field is validated to ensure correct input. The date of birth field must not be less than January 1, 1900, must not be greater than the current time, and must not be greater than the date of the survey. If any of these conditions is not met, a warning dialog is displayed and the invalid input erased. The example assumes a form exists with that has the following fields: DOB (Date) and SurveyDate (Date). The code below would appear in the AFTER section of either DOB or SurveyDate, depending on whichever one will be filled in last.

```
IF (DOB < 01/01/1900) OR (DOB > SYSTEMDATE) OR (DOB > SurveyDate) THEN
   BEEP
   DIALOG "Warning: Invalid date of birth detected"
   CLEAR DOB
END-IF
```

Example 4: An IF command is used to check if a field has been left blank. The example assumes a form exists with the following field: LastName (Text). The code below would appear in the AFTER section of the LastName field.

```
IF NOT LastName = (.) THEN
   BEEP
   DIALOG "Last name field should not be blank."
END-IF
```

Example 5: Multiple IF commands are used to generate more than two possible outcomes. The example assumes a form exists with the following fields: AgeType (Text), AgeYears (Numeric), and Age (Numeric). The code below would appear in the AFTER section of AgeType or Age, depending on which one will be filled in last.

```
IF AgeType = "Days" THEN
   ASSIGN AgeYears = Age / 365.25
END-IF

IF AgeType = "Months" THEN
   ASSIGN AgeYears = Age / 12
END-IF

IF AgeType = "Years" THEN
   ASSIGN AgeYears = Age
END-IF
```

Example 6: The AND operator requires both Sex to be "F" and Pregnancy to be true in order for the GOTO command to be executed. The example assumes a form exists with the following fields: Sex (Text), Pregnancy (Yes/No), and ChildBirth (Yes/No). The code below would appear in the AFTER section of either Sex or Pregnancy, depending on which one is filled in last.

```
IF (Sex = "F") AND (Pregnancy = (+)) THEN
   GOTO ChildBirth
END-IF
```

Example 7: Several IF commands are used to determine if a patient is ill. If any one of the symptoms listed in the form are true, the field ill is assigned true. The example assumes a form exists that has the following fields: Ill (Yes/No), Vomiting (Yes/No), Fever (Yes/No), and Diarrhea (Yes/No). The code below would appear in the AFTER section of the ill field.

```
ASSIGN Ill = (-)
IF Vomiting = (+) THEN
   ASSIGN Ill = (+)
END-IF

IF Diarrhea = (+) THEN
   ASSIGN Ill = (+)
END-IF

IF Fever = (+) THEN
   ASSIGN Ill = (+)
END-IF
```

Example 8: Several IF commands are used to determine the number of symptoms a patient is presenting with. If the number of symptoms is greater than or equal to two, the Case variable is assigned true. If the number of symptoms is less than two, the Case variable is assigned false. The example assumes a form exists that has the following fields: MajorSymp (Numeric), Vomiting (Yes/No), Fever (Yes/No), Diarrhea (Yes/No), and Case (Yes/No).

```
ASSIGN MajorSymp = 0
IF Diarrhea = (+) THEN
   ASSIGN MajorSymp = MajorSymp + 1
END-IF

IF Fever = (+) THEN
   ASSIGN MajorSymp = MajorSymp + 1
END-IF

IF Vomiting = (+) THEN
   ASSIGN MajorSymp = MajorSymp + 1
END-IF

IF MajorSymp >= 2 THEN
   ASSIGN Case = (+)
ELSE
   ASSIGN Case = (-)
END-IF
```

NEWRECORD

Description

This command saves the current records data and opens a new record for data entry.

Syntax

```
NEWRECORD
Example

DIALOG "This is the last field in my form."
NEWRECORD
```

Analysis Commands

ASSIGN

Description

This command assigns numeric or string expression results to a variable. It may be a database variable in a form or data table, or a user-defined variable created by the DEFINE command in a program.

Syntax

```
ASSIGN <variable> = <expression>
LET <variable> = <expression>
```
(ASSIGN and LET may be omitted)

<variable> = <expression>

- The <variable> represents a variable in a database or a defined variable created in a program.
- The <expression> represents any valid arithmetic or string expression.

Program Specific Feature

If the right side of the assignment does not contain a field variable (one in a database table), or a variable that depends on a field variable, the assignment is made immediately.

```
DEFINE YEAR NUMERIC

ASSIGN YEAR = 2000
```

The following code contains two view variables, ONSETDATE and EXPOSUREDATE.

```
DEFINE INCUBATION NUMERIC

ASSIGN INCUBATION = ONSETDATE-EXPOSUREDATE
```

In this example, INCUBATION is only calculated during current dataset processing. It is calculated for each record and may be used similar to a dataset variable in procedures (i.e., TABLES, FREQ, and GRAPH). Prior to and after processing a dataset, INCUBATION will have a "missing" value, although it could be assigned a value with another statement (e.g., INCUBATION = 999).

The value is calculated each time a record that meets the conditions of SELECT is read from the dataset. Any legal expression can be used that combines functions or literal values and operators (i.e., &, +, -, *, /, ^, and MOD). Boolean expressions are not supported in assign commands. Standard variables that depend on database fields must be saved to a table using WRITE before they can be edited using LIST UPDATE.

Comments

Temporary variables must be defined before being used and will accept any type of data (i.e., text, numeric, or date). Once they have been assigned a non-missing value or an expression, their type cannot change.

If an attempt is made to assign an invalid expression to a variable, it retains any previous assignment.

Examples

Example 1: The ASSIGN command is used to assign values to defined variables and database variables. Note that literal values (e.g., 42, other variables, and functions) can be used on the right side of the = operator.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Surveillance
DEFINE State TEXTINPUT
ASSIGN City = "Atlanta"
ASSIGN State = "GA"
ASSIGN Address = City & ", " & State
DEFINE Now DATEFORMAT
ASSIGN Now = SYSTEMTIME
DEFINE Duration NUMERIC
ASSIGN Duration = YEARS(01/01/1998, ReportDate)
DEFINE Ill YN
ASSIGN Ill = (-)
ASSIGN Age = 42
ASSIGN Occupation = "Doctor"
LIST Address City State Duration Now Ill Occupation Age GRIDTABLE
```

BEEP

Description

This command generates a sound.

Syntax

```
BEEP
```

Example

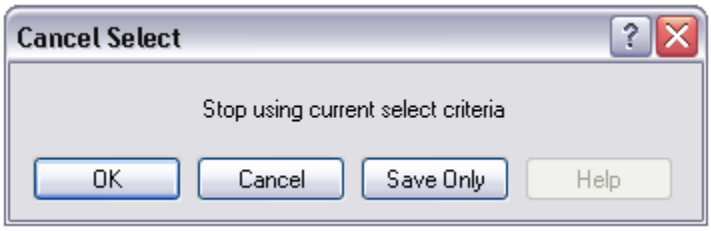
If the number of records in the database is greater than 1,000, a beep is generated and a dialog box appears.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Surveillance
IF RECORDCOUNT > 4 THEN
   BEEP
   DIALOG "Database greater than 4 records."
END
```

CANCEL SELECT or SORT

Description

This command cancels a previous SELECT or SORT command.



Syntax

```
SORT
SELECT
```

Comments

The Cancel Select and Cancel Sort commands automatically close the current output file.

Example

The commands below should be run one-by-one to better understand how the cancel sort and cancel select commands function.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SELECT Ill = (+)
LIST Age Ill Sex
SELECT
SORT Age
LIST Age Ill Sex
SORT
```

CLOSEOUT

Description

This command closes the current output file. It is normally used after a ROUTEOUT command when all the information to be included in the ROUTEOUT file has been produced.

Syntax

```
CLOSEOUT
```

Example

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
ROUTEOUT "C:\ My_Project_Folder \Outbreak1.htm" REPLACE
TABLES Vanilla Ill STRATAVAR=Sex
MEANS Age Ill
CLOSEOUT
```

COXPH

Description

This command performs Cox-Proportional Hazards and Extended Cox-Proportional Hazards survival analysis. This form of survival analysis relates covariates to failure through hazard ratios. A covariate with a hazard ratio greater than one causes failure. A covariate with a hazard ratio less than one improves survival. Some of the subjects may be unavailable prior to failure; the term "censored" is applied to them. COXPH is especially constructed to deal with this situation. Statistics showing the risk set by group and time can be written to an OUTTABLE for later formatting.

Syntax

```
COXPH <time variable>= <covariate(s)>[: <time function>:]  *  <censor variable>
(<value>) [TIMEUNIT="<time unit>"] [OUTTABLE=<tablename>] [GRAPHTYPE="<graph type>"]
[WEIGHTVAR=<weight variable>] [STRATAVAR=<strata variable(s)>] [GRAPH=<graph
variable(s)>]
```

- The <time variable> represents a numeric or date variable, specifying when failure or censorship occurred.
- The <covariate(s)> represent a numeric variable, a non-numeric variable, or a variable specified as non-numeric by parenthesis. Any non-numeric variable, even a variable specified as non-numeric by surrounding with parenthesis, is automatically recoded into dummy variables. For all but one of the levels of a variable, a dummy variable will be created. It measures the contribution of its level to the excluded level. A covariate may be followed by a time function. This causes COXPH to run the Extended Cox procedure.
- The <time function> represents a numeric expression involving the time variable.
- The <censor variable> indicates whether the event is a failure or a censor.
- The <value> indicates which value of the CensorVar represents failure.
- The <strata variable(s)> represents a list of variables indicating the different levels of strata.
- The <weight variable> represents a variable to specify the contribution each data row has on the output.
- The <time unit> represents a value for labeling the time axis.
- The <tablename> represents a valid table name.

The <graph type> generates one of the indicated graphs:

1. Survival Probability shows the adjusted survival curves.

2. Observed shows the observed survival curves.
3. Survival-Observed shows the adjusted and observed survival curves.
4. Log-log Survival shows the logarithm of the negative of the logarithm of the adjusted survival curve.
5. Log-log Observed shows the logarithm of the negative of the logarithm of the observed survival curve.
6. Hazard Function shows the adjusted hazard function.
7. None

- The <graph variable(s)> represent a list of variables used to generate survival curves. Graph variables that are covariates or strata variables create curves adjusted by the covariates at all possible combinations of these graph variables. If a variable is numeric, it is plotted at its average value. Otherwise the graph variable splits the data into separate groups, each with its own curve.

Comments

COXPH uses the Breslow method to handle ties in the data.

Example

In this example, we will use the Anderson dataset from a clinical trial of leukemia patients to compare the treatment and placebo group survival.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Anderson
COXPH STIME = (Rx) * Status ( 1 ) TIMEUNIT="Weeks" PVALUE=95% GRAPH=Rx
GRAPHTYPE="Survival Probability"
```

DEFINE

Description

This command creates a new variable.

Syntax

```
DEFINE <variable> (<scope>) (<type indicator>)
```

- <variable> represents the name of the variable to be created.
- <scope> is the level of visibility and availability of the new variable and must be one of the reserved words STANDARD, GLOBAL, or PERMANENT. If omitted, STANDARD is assumed and a type indicator may not be used.
- <type indicator> **is required** and cannot be used if <scope> is omitted. <type indicator> is the data type of the new variable and must be one of the following reserved words: NUMERIC, TEXTINPUT, YN, or DATEFORMAT. If omitted, the variable type will be inferred based on the data type of the first value assigned to the variable. However, omitting field type **is not** recommended. Once field type is defined, the variable type cannot be changed. An error will occur if you attempt to assign data of a different type to the variable.

Comments

A custom variable defined in Epi Info 7 might not have a predefined data type if the <type indicator> is omitted when the variable is defined. If the variable does not have a predefined type and has not been used, the variable will accept a value in any of the four data types (Text, Number, Date, Yes/No [Boolean]) that is assigned to the variable the first time. Thereafter, the variable takes on the data type of the value assigned and it's data type cannot be changed. However, omitting field type is not recommended. An error will occur if you attempt to assign data of a different data type. Various functions can be used to manipulate data, changing data type of values to match the data type of the variable. Some of these functions include FORMAT, TXTTONUM, TXTTODATE, and NUMTODATE.

Variable Scope

- **STANDARD** variables retain their value only within the current record and are reset when a new record is loaded. Standard variables are used temporarily behaving like other fields in the database. In Classic Analysis, Standard variables lose their values and definitions with each READ statement.
- **GLOBAL** variables retain values across related forms and when the program opens a new form, but are removed when the Classic Analysis program is closed. Global variables persist during program execution. Global variables are also used to store values between changes of data source (e.g., when the READ command is used). Global variables in Classic Analysis may not depend directly or indirectly on table fields.

- **PERMANENT** variables are stored in the EpiInfo.Config.xml file and retain any value assigned until the value is changed by another assignment or the variable is undefined. Permanent variables are shared among Epi Info 7 programs (i.e., Menu, Enter, Classic Analysis, etc.) and persist even if the computer shuts down. Permanent variables in Classic Analysis may not have values that depend directly or indirectly on table fields. A <prompt/description> created for a permanent variable will exist for one session and must be re-established each time it is used.

Type Indicators

- **TEXTINPUT** - Variables of this data type can receive any alpha-numeric characters including symbols and the output of functions (e.g., FORMAT).
- **NUMERIC** - Variables of this data type can receive numbers and the output of functions (e.g., TXTTONUM).
- **DATEFORMAT** - Variables of this data type can receive date values including the output of functions (e.g., TXTTODATE and NUMTODATE).
- **YN** - Variables of this data type can receive the Boolean values of (+) for Yes and (-) for No. Until an assignment is made, YN type variable values are (.) or missing.

Example

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Surveillance
DEFINE Birthday DATEFORMAT
ASSIGN Birthday = 01/01/2006
DEFINE HospitalCode NUMERIC
ASSIGN HospitalCode = 854
DEFINE Smoke YN
ASSIGN Smoke = (+)
LIST Birthday HospitalCode Smoke
```

DEFINE DLLOBJECT

Description

This command allows you to create an ActiveX (dynamic link library or executable) object.

Syntax

```
DLL File

DEFINE <variable> DLLOBJECT "<ActiveX name>.<class>"
WSC File

DEFINE <variable> DLLOBJECT "<filename>"
```

- The <variable> represents the name of the variable to be created.
- The <ActiveX name> represents the internal name of the ActiveX object that contains the class object for DLL files.
- The <class> represents a class name defined within the DLL.
- The <filename> represents the name of the WSC file where the script component resides.

Comments

The ActiveX name may not be the same as the actual name of the dll (dynamic link library) or executable. When an ActiveX object is developed, it is given a project name. This name is required to create the object. The ActiveX object (executable or dll) must be registered before it can be used. Windows Scripting Component (WSC) objects can also be used.

Example

```
DEFINE Week DLLOBJECT "EIEpiwk.Epiweek"
```

Define Group Command (Analysis Reference)

Description

This command allows you to create temporary group variables

Syntax

```
DEFINE <variable> GROUPVAR <variable 1> [<variable 1> ...]
```

- <variable> represents the new group variable being defined.
- <variable 1> ... represent the existing variables to which the new group variable will be equivalent.

Program Specific Feature

Field variables, not defined variables must be in the group. Variables in the group may be from different pages of the same or different forms, in any combination. They may themselves be group variables, but each variable will be represented in the new group only once no matter how many times it appears in the variable list or in group variables in the variable list.

If the group variable being defined exists and is not a group variable, an error message is displayed and the command is not processed. If it is a group variable, the new variable list replaces the existing variable list until the next READ or MERGE command or until it is redefined again. Group variables cannot be undefined.

Comments

This command is useful when there are many variables so that the use of * is impractical or invalid. Group variables can be used in the LIST and WRITE commands and in some statistical commands (e.g., FREQ, TABLES, and MEANS). Group variables cannot be used in Complex Sample commands.

Example

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE Exposure GROUPVAR JELLO CAKES VANILLA CHOCOLATE
LIST Exposure
```

DELETE FILE/TABLES

Description

This command deletes files, tables, and forms.

Syntax

```
DELETE filename | wildcard {RUNSILENT}
DELETE TABLES tablename | filename:tablename {RUNSILENT | SAVEDATA}
```

Comments

Delete file specifies explicitly or implicitly (via wildcards) files to be deleted. If no files are specified, or if some of the specified files cannot be deleted, a message is produced unless RUNSILENT is specified. Wildcards are not permitted in the suffix.

Delete table specifies a table to be deleted. If the table does not exist or cannot be deleted, a message is produced unless RUNSILENT is specified. Unless RUNSILENT is specified, a confirmatory message is displayed prior to deletion.

To delete tables with spaces in their names, specify both the file and the table even if the table is in the current project. The table must be enclosed in single quotes. You can read a table with a space in its name, delete it, causing errors during subsequent procedures. To delete a table with space in its name, specify the database even if the table is in the current database.

DELETE TABLES will not delete, and produce a message if any of the specified tables are data, grid, or program. Code tables are deleted only if they are not referenced by any view.

SAVEDATA does not delete the data tables, but removes the RECStatus property from the table.

Space is not reclaimed until the Compact utility is run.

Analysis will not delete the current project MDB or any table in use by the current form.

Example

```
DELETE TABLES Backup2005
```

DELETE RECORDS

Description

This command deletes selected records.

Syntax

```
DELETE * {PERMANENT | RUNSILENT}
DELETE expression {PERMANENT | RUNSILENT}
```

Comments

All records in the current selection are set to deleted status, or if PERMANENT is specified, physically deleted. A confirmation message is displayed unless RUNSILENT is selected. This applies to Access tables and may not be used when using related tables. Use Epi Info 3.5.3 Compact Database utility to reclaim space after deleting tables.

Examples

Example 1: The code below will permanently erase all records in the current data source where the AgeInDays variable contains a value greater than five. Permanent deletions cannot be undone.

```
DELETE AgeInDays > 5 PERMANENT
```

Example 2: The code below will mark all records in an Epi Info 7 database for deletion. The UNDELETE Command can restore records that have been marked for deletion.

```
DELETE *
```

DIALOG

How to Use the DIALOG Command

Description

This command provides interaction with the user from within a program. Dialogs can display information, ask for and receive input, and offer lists for making choices.

Syntax

Simple Form

```
DIALOG "<prompt>" {TITLETEXT="<title>"}
```

- <prompt> represents the text to be displayed as message.

- <title> represents the text to be used as the caption for the dialog window. If <title> is omitted in the Dialog command, "Analysis" will be displayed on the dialog box's title bar.

Get Variable Form

```
DIALOG "<prompt>" <variable> <entry type> {TITLETEXT="<title>"}
DIALOG "<prompt>" <variable> "<value 1>", "<value 2>", "<value 3>", … ,"<value n>"
{TITLETEXT="<title>"}
DIALOG "<file type selection>" <variable> READ {TITLETEXT="<title>"}
DIALOG "<file type selection>" <variable> WRITE {TITLETEXT="<title>"}
```

- <variable> represents the variable to store value entered.
- <entry type> represents a reserved word and/or mask that defines the type of input to be accepted and stored in the variable. The following are valid entry types:
  - **TEXTINPUT** specifies the input as text.
  - **YN** specifies the input as Boolean.
  - **DATEFORMAT** ("<date mask>") specifies the input as a date; if a date mask is not specified, the system short date is used.
  - "<numeric mask>" specifies the input as numeric. This option is identified as "Number Only". The numeric mask should be a text string (e.g., "####" or "##.###") that indicates the type of data to be entered. # indicates a digit.
  - If no <entry type> is specified, the input variable is interpreted as a number if possible. If the value is not a valid number, but is a valid date, it is interpreted as a date. Otherwise, an error occurs.
- <value> represents a value in a drop-down list of choices. Each value included in the command will be shown as a single item in the list.
- <file type selection> controls the type of files that are displayed for selection. It consists of alternating description and filter elements separated by vertical bars. The description is displayed for you to select. The corresponding filter element selects the files. If this element is left blank, all files can be selected. Syntax is created using the File Open and File Save options from the Dialog Format drop down menu.

List of Values Form

```
DIALOG "<prompt>" <variable> [<list type>] {TITLETEXT="<title>"}
DIALOG "<prompt>" <variable> DBVALUES <table name> {TITLETEXT="<title>"}
```

- The [<list type>] are DATABASES and DBVIEWS

Comments

The Simple form of DIALOG places a dialog box on the screen, using the text provided, with an OK button.

The Get Variable form of DIALOG displays the text prompt and provides a means for entering a value.

- If **YN** is specified, "Yes," "No" and "Cancel" buttons are presented and the specified variable is set to (+), (-), or (.).
- If **TEXTINPUT** or no entry type is specified, an entry field for user response is provided with OK and Cancel buttons. The variable is assigned the value of the input with a missing value if Cancel is chosen.
- The **Get Variable** form of the DIALOG command can also display a file selection dialog. If READ is used, only existing files are displayed. If WRITE is used and an existing file is selected, you will see the Overwrite? prompt.
- The **Multiple Choice** form of DIALOG displays the text prompt and provides a combo box for selecting among the values with OK and Cancel buttons. The variable is assigned the value of the input with a missing value if Cancel is chosen. Variable will text type.
- The **Date** form of the DIALOG command uses a special control for accepting input. Each section of the date (year, month, and day) can be increased or decreased independently using the drop down calendar. Using this control, it is impossible to select an invalid date.

The List of Values form of DIALOG displays a combo box of databases or form variables with OK and Cancel presented. The variable is assigned the value of your input with a missing value if Cancel is chosen. Variable will be text type. The VARIABLE VALUE form of DIALOG displays a combo box of distinct values of the specified variable in the specified database. The variable is assigned the value of input with a missing value if Cancel is chosen. The variable will be of the same type as the database variable.

Date Mask	
*	System Time Format
!	System Long Date

Numeric Mask	
#	Digit placeholder (Entry required).
.	Decimal placeholder. The actual character used is the one specified as the decimal placeholder in the computer's international settings.
,	Thousand separator. The actual character used is the one specified as the thousands

Numeric Mask	
	placeholder in the computer's international settings.
9	Digit placeholder. (Entry optional).
Punctuation	Included in the display.

Custom Formats	
D	One- or two-digit day.
Dd	Two-digit day. Single-digit day values are preceded by a zero.
Ddd	Three-character weekday abbreviation.
dddd	Full weekday name.
H	One- or two-digit in a 12- hour format.
Hh	Two-digit hour in a 12-hour format. Single digit values are preceded by a zero.
H	One- or two-digit hour in 24-hour format.
HH	Two-digit hour in a 24-hour format. Single digit values are preceded by a zero.
M	One- or two-digit minute.
Mm	Two-digit minute. Single digit values are preceded by a zero.
M	One- or two-digit month number.
MM	Two-digit month number. Single digit values are preceded by a zero.
MMM	Three-character month abbreviation.
MMMM	Full month name.

Custom Formats	
S	One- or two-digit seconds.
Ss	Two-digit seconds. Single- digit day values are preceded by a zero.
T	One-letter AM/PM abbreviation. AM displays as A.
Tt	Two-letter AM/PM abbreviation. AM displays as AM.
Y	One-digit year. 1997 displays as 7.
Yy	Last two digits of the year. 1997 displays as 97.
Yyyy	Full year. 1997 displays as 1997.
Punctuation	Included in the display

Examples

Example 1: A prompt is displayed letting you know that the following commands may take several minutes to complete because of their complexity.

```
DIALOG "Warning: This script may take several minutes to complete" TITLETEXT="Warning"
```

Example 2: The DIALOG command is used to obtain a user-supplied date to calculate the patient's age. If you do not supply a date (press the Cancel button), the default ending date contained in the survey data source is used instead.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Surveillance
DEFINE UpdateDate DATEFORMAT
DEFINE SubmitDate Dateformat
ASSIGN SubmitDate = EventDate
DIALOG "Enter a new ending date for survey data:" UpdateDate DATEFORMAT "MM-DD-YYYY"
DEFINE NewAge NUMERIC
ASSIGN NewAge = YEARS(SubmitDate, UpdateDate)
GRAPH NewAge GRAPHTYPE="Column"
```

Example 3: You will see a drop-down list of choices. The list contains part of the command; in this case, several counties in the state of Georgia. Your selection is temporarily assigned to all records in the form.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
```

```
DEFINE NewCounty TEXTINPUT
DIALOG "Select a county" NewCounty "Fulton", "Baldwin", "DeKalb", "Cobb"
ASSIGN CountyName = NewCounty
LIST CountyName
```

Example 4: A drop-down list of choices is displayed. Each list item is retrieved from the X_COORD column of the SohoDead data table.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}::SohoDead
DEFINE Coordinates TEXTINPUT
DIALOG "Select Coordinates" Coordinates DBVALUES SohoDead X_COORD
TITLETEXT="Coordinates"
```

Example 5: A dialog box prompts you for an image file in JPG or bitmap format. Upon choosing the file, the full path and filename of the file are saved to the specified variable.

```
DEFINE FileName TEXTINPUT
DIALOG "Image files|*.bmp;*.jpg;|Allfiles|*.*" FileName READ TITLETEXT="Get image
files"
```

Example 6: A dialog box is displayed that allows you to enter a number. An input mask is used to force your input to three whole digits and one decimal digit. Your input is temporarily assigned to each record in the form.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE NewAge NUMERIC
DIALOG "Enter patient age" NewAge "###.#"
ASSIGN Age = NewAge
LIST Age
```

Example 7: A dialog box is displayed that allows you to enter a number. An input mask is used in order to force your input to either one or two whole digits and one decimal digit. (The 9 represents an optional digit.) Your input is temporarily assigned to each record in the form.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE NewAge NUMERIC
DIALOG "Enter patient age" NewAge "9##.#"
ASSIGN Age = NewAge
LIST Age
```

DISPLAY

Command Generator

How to Use the DISPLAY Command

Description

This command displays table, form, and database information.

Syntax

```
DISPLAY <option> [<sub-option>] [OUTTABLE=<tablename>]
```

- The <option> determines the type and source of information displayed.
- The <tablename> represents the name of the data output table to receive results (optional).
- The [<sub-option>] refers to the type of displayed information.

    1. **Option DBVARIABLES** - displays information about variables. Sub-option DEFINE displays only defined variables. Sub-option FIELDVAR displays only table/view variables for the current READ and RELATE tables. Sub-option LIST followed by a list of variable names displays only the specified variables.
    2. **Option DBVIEWS** - displays information about forms and other Epi Info 7 specific tables in a database. The sub-option is the path of the database. Omitting the sub-option displays information for the current project database.
    3. **Option TABLES** - displays information about tables in a database, whether Epi Info 7 specific or generic. The sub-option is the path of the database. Omitting the sub-option displays information for the current project database.

Examples

Example 1: The DISPLAY command is used to show the tables, forms, and variables from an existing data source.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DISPLAY TABLES
DISPLAY DBVIEWS
DISPLAY DBVARIABLES
```

Example 2: All forms in the Sample database are displayed.

```
DISPLAY DBVIEWS 'C:\Epi_Info_7\Projects\Sample\Sample.prj'
```

Example 3: All tables in the Sample database are displayed.

```
DISPLAY TABLES 'C:\Epi_Info_7\Projects\Sample\Sample.prj'
```

Example 4: All variables in Oswego are written to a new table in the Sample database called VarInfo.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DISPLAY DBVARIABLES OUTTABLE=VarInfo
```

EXECUTE

Description

This command executes a Windows program; one explicitly named in the command, or one designated within the Windows registry as appropriate for a document with a named file extension. This provides a mechanism for bringing up whatever word processor or browser is the default on a computer without first knowing its name.

If the pathname is a long filename, it must be surrounded in single quotes. If the command takes parameters, surround the command and the parameters with a single set of double quotes. Do not use single quotes.

Program Specific Feature

Link and Activate are not valid command names.

Syntax

```
EXECUTE <filename>
EXECUTE "<filename> <command-line parameters>"
EXECUTE NOWAITFOREXIT <filename>
EXECUTE NOWAITFOREXIT '<filename>'
EXECUTE NOWAITFOREXIT "<filename>"
EXECUTE WAITFOREXIT <filename>
EXECUTE WAITFOREXIT '<filename>'
EXECUTE WAITFOREXIT "<filename>"
```

- The <filename> represents the path and program name for .exe (filename for registered Windows programs) and .com (filename for MS-DOS binary executable) files.
- The <command-line parameters> represent any additional command-line arguments that the program can accept. When used, the entire string should be enclosed within double quotes.
- When Wait for Command to Execute (modal) is specified, the command and Analysis should run. When Wait for Command to Execute is not specified (non-modal), Classic Analysis should wait until the executed program closes before continuing. When EXECUTE is run modally, permanent variables are written before the command is executed, and reloaded after the command is executed.

Comments

If the name given is not a program, but a file with an extension (the three characters after the ".") registered by Windows for displaying the document, the correct program to display the file will be activated (i.e., WRITEUP.DOC might cause Microsoft Word © to run and load the file on one computer). On another machine, however, .DOC might correspond to Corel WordPerfect©. Usually .TXT will run NOTEPAD.EXE© or WORDPAD.EXE©, and

image files will appear in a browser or in a graphics program. An .HTM file will bring up the default browser.

Examples

Example 1: The EXECUTE command is used to start the Enter module. The command-line parameter is used to load the Oswego Form from the Sample database.

```
EXECUTE "C:\Epi Info 7\
Epi Info 7\Enter.exe"
```

Example 2: The output file generated (in this case, Outbreak1.htm) is opened in the computer's default web browser.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
ROUTEOUT "C:\Epi Info 7\Epi Info 7\Output\Outbreak1.htm' REPLACE
SET STATISTICS=COMPLETE
TABLES Vanilla Ill STRATAVAR=Sex
MEANS Chocolate Ill
CLOSEOUT
EXECUTE "C:\Epi Info 7\Epi Info 7\Outbreak1.htm"
```

FREQ

Description

FREQ produces a table from the table(s) specified in the last READ statement, showing how many records have each value of the variable. Exact Confidence limits for each proportion are included.

Syntax

```
FREQ [<variable(s)>] {<settings>}

FREQ * {EXCEPT [<variable(s)>]} {<settings>}
```

- <variable(s)> represents one or more variable names. Group variables may be used.
- <settings> represent clauses from the SET command indicating a value of a setting (except PROCESS and HYPERLINKS) which will be used for the duration of the statistical command only.

Comments

Records may be included or excluded from the count by using SELECT statements. Those marked as deleted in Enter will be handled according to the current setting for SET PROCESS. If more than one variable name is given, FREQ makes a separate table for each variable. Confidence limits for the binomial proportions are produced.

If a WEIGHTVAR is specified, the value of the WEIGHTVAR variable is treated as a count of instances of the variable being computed in the frequency (i.e., in the following command a record containing the value 30 for AGE and 15 for COUNT would give a result equivalent to 15 individuals of age 30).

FREQ AGE WEIGHTVAR = COUNT

If STRATUM is specified, a separate frequency is produced for each stratifying variable value.

FREQ ILL STRATAVAR=SEX, produces a table showing ILL (Yes/No/Unknown) for males and another for females. The same numbers can be obtained using TABLES ILL SEX, but the latter produces results in one table rather than in separate tables, and produces statistics to test for an association between ILL and SEX.

FREQ * makes a table for each variable in the current form other than unique identifiers. It is often used to begin analyses of a new data set.

To do frequencies of all variables except a few, use FREQ * EXCEPT VarName(s) followed by the names of the variables to be excluded.

Multiline (memo) variables cannot be used in Frequencies. To use a Multiline variable, define a new variable and assign to it the value SUBSTRING(<old variable>,1,255) and use it in the frequency.

Examples

Example 1: The number of ill and healthy people are displayed along with their percentages and the total.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ Ill
```

Example 2: In this case, the variable 'Desserts' is a group variable containing the Yes/No variables Chocolate, Vanilla, and Cakes. Running a frequency on a group variable automatically runs a frequency on every variable contained in the group.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ Desserts
```

Example 3: A frequency on two variables is produced. In this case, BakedHam and Milk.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ BakedHam Milk
```

Example 4: A frequency on every variable in the current data source is produced.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ *
```

Example 5: A frequency on every variable in the current data source (except Age, Code_RW, DateOnset, Name and TimeSupper) is produced.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ * EXCEPT Age Code_RW DateOnset Name TimeSupper
```

Example 6: A frequency of ill people is produced, stratified by sex. Using the stratification option will produce two frequencies. In this case, males and females.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ Ill STRATAVAR=Sex
```

Example 7: A weighted frequency is conducted. For each record, the value stored in Count is used to represent that record's weight.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Lasum
FREQ Outcome WEIGHTVAR=Count
```

Example 8: A complex sample frequency is run using the Epi1 data set.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Epi1
SET STATISTICS=COMPLETE
FREQ VAC PSUVAR=Cluster
```

GRAPH

Description

This command produces graphs from data in Classic Analysis.

Syntax

```
GRAPH [<variable(s)>] GRAPHTYPE="<graph type>" [<option name>=option value]

GRAPH [<variable(s)>] * <crosstab> GRAPHTYPE="<graph type>"

GRAPH [<variable(s)>] GRAPHTYPE="<graph type>" XTITLE="<string>" YTITLE="<string>"
```

- The <variable(s)> represents the (main) variable(s) to be graphed.
- The <crosstab> represents a variable to be used to classify the main variable(s).
- The <graph type> represents one of the graph types described below.
- **STRATAVAR=<variable>** generates a graph for each value of VarName. When saving a template, the template stores the current graph's background including the main title and the subtitle if it exists. However, if the sub-title is stratified (under "One Graph for Each Value of" in the dialog screen), it will be filtered out.
- **WEIGHTVAR=<variable>** weights each record by the value of <VarName>.
- **WEIGHTVAR=<agg func>(<variable>)** allows multiple records referring to the same values of the main variable, crosstab variable (if any), and strata variable (if any) are represented by the aggregate of VarName. Permissible AggFunc values are MIN (minimum), MAX (maximum), AVG (average), STDEV (standard deviation), SUM, SUMPCT (percent based on total of VarName), COUNT (number of records), or PERCENT (percent based on number of records).
- **TITLETEXT="<string>"** represents a title for each page of graphs. This title is in addition to the title for each graph, which is set by customization.
- **DATEFORMAT="<format string>"** is used when a main variable is a date variable to determine the format in which it will be displayed. Uses the same coding scheme as the FORMAT function.
- **XTITLE="<string>", YTITLE="<string>"** are used to pass X- and Y-axis labels from the GRAPH command.

Program Specific Feature

Multiline (memo) fields cannot be graphed. To use a Multiline variable, define a new variable, assign to it the value SUBSTRING(<old variable>,1,255), and use it in the graph.

The following are the graph types capable of being generated by the GRAPH command along with type-specific information ("series" refers to the values of a main variable for a specific value of any crosstab and strata variables):

- **Line graphs** connect X-Y points with lines. The main variables are the X and Y variables. Each series is represented by a different style of line. Both variables must be numeric. To generate a line graph with categorical data, generate a Bar graph and use customization.
- **Column graphs** use vertical bars to represent the count or weight for each value of the main variable(s). Each series results in an additional vertical bar at each point; the bars are distinguished by their style.
- **Bar graphs** use horizontal bars to represent the count or weight of each value of the main variable(s). Each series creates an additional horizontal bar at each point.
- **Epi Curve** use vertical bars to represent the count or weight for each value of the main variable. Each series creates an additional vertical bar at each point. The main variable must be numeric. This graph differs from the bar graph because adjacent bars represent equal ranges of the main variable.
- **Pie** in which each series is represented by a circle, and each value of the main variable has a slice of the circle proportional to the value associated with it.
- **Area** is similar to a line, except that the area below the line contains a solid fill.
- **Scatter graphs** display X-Y points as a scatter gram. The main variables are X and Y variables. Each series is represented by a different style of point.
- **Bubble graphs** are a variation of a Scatter chart in which the data points are replaced with bubbles. A Bubble chart can be used instead of a Scatter chart if your data has three data series.

Examples

Example 1: A graph showing the age of all survey respondents is displayed.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
GRAPH Age GraphType="Column"
```

Example 2: A pie chart showing age categories is displayed.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE AgeCategory TEXTINPUT

RECODE Age TO AgeCategory

LOVALUE - 0 = "Unknown"

0 - 10 = "0 - 10"

10 - 20 = "11 - 20"

20 - 30 = "21 - 30"

30 - 50 = "31 - 50"

50 - 70 = "51 - 70"
```

```
70 - HIVALUE = ">70"
END

GRAPH AgeCategory GRAPHTYPE="Pie" TITLETEXT="Church Supper Attendees"
```

Example 3: A scatter graph is displayed showing age by time of supper.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
GRAPH TimeSupper Age GRAPHTYPE="Scatter XY"
```

Example 4: A graph showing the number of ill persons and vanilla eaters is displayed using the form for the Oswego outbreak investigation.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
FREQ Vanilla Ill STRATAVAR=Sex OUTTABLE=VanillaOut
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:VanillaOut
SELECT Vanilla = (+) OR Ill = (+)
SET Missing = (-)
DEFINE Vanilla2  YN
DEFINE Ill2 YN
IF Ill = (+) THEN
   ASSIGN Ill2 = Sex
END
IF Vanilla = (+) THEN
   ASSIGN Vanilla2 = Sex
END
GRAPH Ill2 Vanilla2 GRAPHTYPE="Bar" TITLETEXT="Number of Ill Persons and Vanilla Eate
rs by Sex" WEIGHTVAR=Count
```

Example 5: An Epi Curve showing the incubation time for an unknown pathogen is displayed. Note that the DIALOG=(-) parameter ensures the graph window does not appear. Instead, the graph is generated and sent to the Analysis output window. This option can be useful when running automated scripts since it does not require user interaction to continue processing commands.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE Incubation NUMERIC
ASSIGN Incubation = MINUTES(TimeSupper, DateOnset) / 60
GRAPH Incubation GRAPHTYPE="Epi Curve" XTITLE="Incubation Period (Hours)"
YTITLE="Number of Cases"
```

IF THEN ELSE

Description

This command defines conditions and one or more consequences which occur if the conditions are met. An alternative consequence can be given after the ELSE statement. The ELSE will be executed if the first set of conditions is not true. If the condition is true, the first statement is executed. If the statement is false, the statement is bypassed. If an ELSE statement exists, it is executed instead. THEN is part of the If Condition statement. It starts the section of code executed when the If condition is true.

Syntax

```
IF <expression> THEN
   [command(s)]
END

IF <expression> THEN
   [command(s)]
ELSE
   [command(s)]
END
```

- The <expression> represents a condition that determines whether or not subsequent commands will be run. If the condition evaluates to true, the commands inside of the IF block will run. If the condition evaluates to false, the commands inside of the ELSE block will run instead. If no ELSE exists and the condition is false, then no commands inside of the IF block are run.
- The [command(s)] represents at least one valid command.
- The ELSE statement is optional and will run any code contained inside of it when the <expression> evaluates to false.

Comments

An IF statement is executed immediately if it does not refer to a database variable, if characteristic or attribute that can be measured, or if any defined variables have been assigned literal values. If the statement, YEAR = 97 has already occurred, then an IF statement dependent on it, such as IF YEAR = 97 then …., is executed immediately.

```
IF Age > 15 THEN
   ASSIGN Group = "ADULT"
ELSE
   ASSIGN Group = "CHILD"
END
```

It is important to cover all the conditions in IF statements to avoid gaps in the logic and results. Sometimes it is important to have an ELSE condition that covers conditions not included in other IF clauses. This effect is best achieved by setting the variable initially to something other than missing.

```
DEFINE ILL YN
ASSIGN ILL = (-)

IF Vomiting = (+) THEN
```

```
ASSIGN ILL = (+)
END

IF Diarrhea = (+) THEN
   ASSIGN ILL = (+)
END

IF Fever = (+) THEN
   ASSIGN ILL = (+)
END
Set Ill = (+) only if one or more symptoms are present.

The same result could be achieved with this code:

IF (Diarrhea = (+)) OR (Vomiting = (+)) OR (Fever = (+)) THEN
   ASSIGN ILL = (+)
ELSE
   ASSIGN ILL = (-)
END
```

Examples

Example 1: The Group variable for all records in the data set is assigned the value of "Young Adult" if the Age variable has a value between (but not including) 17 and 30. If the value in Age falls outside of this range, no value is assigned to Group.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE Group TEXTINPUT
IF Age < 30 AND Age > 17 THEN
   ASSIGN Group = "Young Adult"
END

LIST Group
```

Example 2: Several different values are assigned to the Group variable depending on the value of the Age variable.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE Group TEXTINPUT
IF Age < 30 AND Age > 17 THEN
   ASSIGN Group = "Young Adult"
END

IF Age <= 17 THEN
   ASSIGN Group = "Minor"
END

IF Age >= 30 THEN
   ASSIGN Group = "Adult"
END

LIST Group
```

Example 3: If the patient ate chocolate or vanilla ice cream, the variable IceCream is assigned a value of true. Otherwise, it is assigned a value of false.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE IceCream YN

IF Vanilla = (+) OR Chocolate = (+) THEN
   ASSIGN IceCream = (+)
ELSE
   ASSIGN IceCream = (-)
```

```
END

LIST Vanilla Chocolate IceCream
```

KMSURVIVAL

Description

This command performs the Kaplan-Meier Survival Analysis and produces graphs and statistics for one or several groups of subjects being followed in a clinical study. At any given time, some of the subjects may be "censored," not having information available on their status. KMSurvival is especially constructed to deal with this situation. Statistics showing the risk set by group and time can be written to an OUTTABLE for later formatting.

Syntax

```
KMSURVIVAL <time variable>=<group variable> * <censor variable> (<value>)
[TIMEUNIT="<time unit>"] [OUTTABLE=<tablename>] [GRAPHTYPE="<graph type>"]
[WEIGHTVAR=<weight variable>]
```

- The <time variable> represents the variable specifying the time of the event.
- The <group variable> represents the variable that indicates to which treatment or control group the subject belongs.
- The <censor variable> represents the variable to describe an event as failure or censored.
- The <value> represents the value of the censor variable indicating uncensored (failure).
- The <time unit> represents a value for labeling the time axis.
- The <graph type> represents the following:
  - **Survival Probability** is the observed survival over the different groups.
  - **Log-Log Survival** is the log of the negative log of the survival probability.
  - **None** does not produce a graph. If no graph type is specified, the default Survival Probability curve is plotted.
- The <weight variable> represents a variable in the database that specifies the contribution or each data row to the output.

Example

The commands below show a comparison between two groups.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Addicts
KMSURVIVAL Survival_Time_Days = Clinic * Status ( 1 ) TIMEUNIT="Days"
```

LIST

Description

This command does a line listing of the current dataset. If variable names are given, LIST displays only these variables. LIST * displays all variables of all active records. LIST * EXCEPT displays all variables of all active records except those named.

Syntax

```
LIST {* EXCEPT} [<variable(s)>] LIST {* EXCEPT} [<variable(s)>] {GRIDTABLE}
```

- The * asterisk is used to represent all variables
- The <variable(s)> represents one or more variable names.
- The keyword GRIDTABLE specifies that data is displayed as a grid for viewing only, instead of HTML format.

Comments

Adding an EXCEPT Variable list excludes all the named variables from a LIST or LIST *.

If the dataset has been sorted with the SORT command, the records are listed in sorted order. Otherwise, they are listed in an order determined by the database.

Examples

Example 1: All variables are listed using the grid format.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
LIST * GRIDTABLE
```

Example 2: The Age variable is listed using the web (HTML) format.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
LIST Age
```

Example 3: The variables Sex, Vanilla, Chocolate, Ill, and Age are listed using the grid format.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
LIST Sex Vanilla Chocolate Ill Age GRIDTABLE
```

LOGISTIC

Description

This command performs conditional or unconditional multivariate logistic regression with automatic dummy variables and support for multiple interactions.

Syntax

```
LOGISTIC <dependent variable> = <independent variable(s)> [MATCHVAR=<match variable>]
[NOINTERCEPT] [OUTTABLE=<tablename>] [WEIGHTVAR=<weight variable>] [PVALUE=<PValue>]
```

- The <dependent variable> represents the dependent variable.
- The <independent variable(s)> represents an independent variable that can be a numeric variable, a non-numeric variable, or a variable surrounded by parenthesis. Any text or yes-no variable, or a variable surrounded with parenthesis, is automatically recoded into dummy variables. A dummy variable is created for all but one of the levels of a variable. The variable measures the contribution of its level relative to the excluded level. Interactions are specified by * between variables.
- The <weight variable> represents a variable to specify the contribution each data row has to the output.
- The <match variable> represents a variable to specify the different groups for a conditional analysis. If a match variable is specified, a conditional analysis will be performed
- The <tablename> represents a table where the residuals are stored. If no table name is present, no residuals are produced.
- The <PValue> represents the size of the confidence intervals; this may be specified as percent or decimal. If greater than .5, 1-PValue is used. The default is 95%.

Comments

LOGISTIC uses the Newton-Rhapson method for maximizing likelihood.

Example

To do an unconditional regression, run the following:

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
LOGISTIC Ill = BrownBread CabbageSal Water Milk Chocolate Vanilla
```

MEANS

Description

This command is used to compute descriptive statistics for a continuous (numeric) variable. When used with a cross-tabulation variable, it also computes statistics showing the likelihood that the means of the groups are equal. The mean of a yes-no variable is the proportion of respondents answering yes.

Syntax

```
MEANS <variable 1> {<variable 2>} {STRATAVAR=<variable(s)>} {WEIGHTVAR=<variable>}
{OUTTABLE=<tablename>}
```

- **<variable 1>** represents a numeric variable to be used to calculate means (or * for all numeric variables).
- **<variable 2>** represents any variable used for cross-tabulation (optional).
- **<variable(s)>** represent variable(s) to be used for stratified analysis.
- **<variable>** represents a variable containing the frequency for the event.
- **<tablename>** represents a name for a table to be created.

Comments

The MEANS command has two formats. If only one variable is supplied, the program produces a table similar to one produced by FREQUENCIES, plus descriptive statistics. If two variables are supplied, the first is a numeric variable containing data to be analyzed and the second is a variable that indicates how groups will be distinguished. The output of this format is a table similar to one produced by TABLES, plus descriptive statistics of the numeric variable for each value of the group variable.

Multiline (memo) variables cannot be used in MEANS. To use a Multiline variable, define a new variable and assign to it the value SUBSTRING(<old variable>,1,255) and use it in the means.

The f-test which is generated from MEANS is a generalization of the t-test. The t-test only works with two groups while the f-test works with any number of groups.

MEANS produces the following statistical tests:

- Parametric
- ANOVA (for two or more samples)
- Student's t-test (for two samples)
- Non-parametric
- Kruskal-Wallis one-way analysis of variance (for two or more samples)
- Mann-Whitney U = Wilcoxon Rank Sum Test (for two samples)

Examples

Example 1: Descriptive statistics for the age variable are displayed, including the number of observations, the total, the mean, variance, standard deviation, 25%, median, 75%, maximum, minimum, and mode.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
MEANS Age
```

Example 2: The MEANS command is used to compare two means. An independent t-test and one-way analysis of variance (ANOVA) is performed.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:EvansCounty
MEANS CHL CHD
```

Example 3: The average number of cigarettes smoked between males and females is determined.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Smoke
MEANS NumCigar Sex STRATAVAR=Strata WEIGHTVAR=SampW PSUVAR=PSUID
```

MERGE

Description

This command merges records in one dataset with those in another the Global Record Identifier contained in every Epi Info 7 project to establish the match between records. Records in the second dataset that do not have matching keys can be appended to the end of a dataset. Records in the second dataset that do have matching keys can be used to update records in the main dataset. Records in the second dataset can be used as the parent in defining an Epi Info 7 parent-child relationship after records have been merged into the parent and child forms.

Syntax

```
MERGE <table specification> {LINKNAME=<text>} [<key(s)>] <type>
```

- The <table specification> represents the type and name of a data table to be read as the Merge file.

  The <key(s)> represent the Global Record Identifier on which the match or relate will be performed. These are in the form: <ExpressionCurrntTable>::<ExpressionMergeTable>

- The <MergeType>, may be APPEND, UPDATE or RELATE. If no type is specified, APPEND and UPDATE are performed. For matching records, UPDATE replaces the value of any field in the READ table whose name matches in the MERGE table with the value from the MERGE table unless it has a missing value. For unmatched records, APPEND creates a new record in the READ table with values only for those fields which exist in the MERGE table.

- Currently, Merge is only supported when the READ and MERGE data source is an Epi Info 7 project.

Comments

APPEND adds unmatched records in the Merge table to the currently active dataset. Only fields found in both datasets are added.

UPDATE replaces fields of records in the active table with those in the Merge table if the key expressions match. Only fields found in both datasets with a non-empty value in the Merge table are replaced.

RELATE moves the unique key of the current table to the foreign key of the related table to make a permanent relationship. The related (Merge) table must be an Access/Epi2000 table. The READ table and the RELATE tables must be Epi Info forms. READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:

Example

```
MERGE {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Surveillance GlobalRecordId ::
GlobalRecordId
```

PRINTOUT

Description

This command sends the contents of the current output file (the one visible in the output window) or some other specified file to the default printer.

Syntax

```
PRINTOUT <filename>
```

Example

The output file Oswego.txt is sent to the printer.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
WRITE REPLACE "Text" 'C:\Epi_Info\Oswego.txt' *
PRINTOUT 'C:\Epi_Info\Oswego.txt'
```

QUIT

Description

This command closes the current data files and terminates the current program, closing Classic Analysis.

Syntax

```
QUIT
```

Program Specific Features

Quit will stop the execution of a program and close Classic Analysis. If there is no QUIT at the end of a .PGM program, Classic Analysis continues to run and offer user-interaction.

Example

You are presented with a dialog box asking if you want to close Analysis. Selecting 'Yes' closes Classic Analysis.

```
DEFINE Results YN
DIALOG "Do you wish to close Analysis?" Results YN
IF Results = (+) THEN
   QUIT
END
```

READ

Description

This command makes one or more forms the active dataset. It also removes any previously active datasets and associated defined variables and dataset-specific commands (e.g., RELATE, SORT, SELECT or IF statements).

Syntax

```
READ <table specification> FMT="<file format>" UID="<username>" PWD="<password>" END
```

The <table specification> and FILESPEC are referred to in the following paragraphs:

The READ, RELATE, and MERGE commands can operate on many different types of data. Each type requires a different table specification, and some types required additional information in a file specification. Table specifications usually consist of a data type (contained in double quotes), a space, a file path (which may be enclosed in single quotes, and must be if it includes a space), a colon, and a table name.

The various file types that can be used are:

- Epi Info 7 Forms and Tables
- Microsoft Access 97-2003 and MS Access 2007
- Microsoft Excel 97-2003 and Excel 2007
- **SQL Server** – Server name and Database name will be required when accessing tables within an SQL Server database.
- **Text Files** – There are two basically different forms of text files. Both forms have only one table per file, so there is no need to specify a table. Both forms put the data for one record on a single line. The difference is in how the fields are indicated. One form, called "delimited," uses designated characters to separate fields. The second form, shown in the fourth example, is called "fixed" because each field occupies the same positions in each line. All character positions through the last field must be accounted for even if they do not contain useful data (the command generator will automatically generate filler fields as required). Even if the first line of the file contains field names (HDR="YES"), the names specified in field definitions will be used. The text file driver actually reads the file into the database, so changes made to the file after the READ will not be saved and changes made through Epi Info will not be saved to the text file (unless it is rewritten with the WRITE REPLACE command, which is available only in CSV format).

Examples

Example 1: If the file/form was previously accessed, the Oswego Form is read

```
READ {config:Sample.prj}:Oswego
```

Example 2: The Oswego Form is read. Unlike in example 1, the full path to the database file is specified.

```
READ {C:\Epi Info 7\Epi Info 7\Projects\Sample\Sample.prj}:Oswego
```

Example 3: If a space appears in the table name in Access, it must be enclosed in square brackets.

```
READ {C:\Epi Info 7\Epi Info 7\MyData}:[Table Name with Spaces]
```

Example 4: An Excel spreadsheet is read.

```
READ {C:\Epi Info 7\Epi Info 7\PlagueData.xls}:[Plague$]
```

RECODE

Description

This command is used to change some or all the values of a variable. New values can be stored in the same variable or in a new one. It can also be used to convert a numeric variable into a character variable or the reverse, or to create a new variable based on recoded values of an existing variable.

Syntax

```
RECODE <variable1> TO <variable2>
   value1 - value2 = <recoded value>
   value1 – HIVALUE = <recoded value>
   LOVALUE - value2 = <recoded value>
   value3 = <recoded value>
ELSE = <recoded value>
END
```

- The <variable1> represents the donor variable (where the values are).
- The <variable2> represents the receiver variable (where recoded values will be).

Comments

Text values must be enclosed in quotation marks; numeric, date; yes/no values must not. All recoded values must be of the same type. Numeric ranges are separated by a space, hyphen, and space, as in 1 - 5. Negative values are permitted (i.e.,-10, -9, and -8). The words LOVALUE and HIVALUE may be used to indicate the smallest and largest values representable in the database. The word ELSE may be used to indicate all values not falling in the preceding ranges. Recodes take place in the order stated; if two ranges overlap, the first in order will apply. Analysis cannot RECODE more than about 12 levels of values. If this is a problem, do as many recodes as possible, write a new table, READ it, and do more recodes.

Examples

Example 1: The RECODE command is used to generate an age range.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE AgeRange TEXTINPUT
RECODE Age TO AgeRange LOVALUE - 0 = "<=0" 0 - 10 = ">0 - 10"= 10 - 20 = ">10 - 20" 20
- 30 = ">20 - 30" 30 - 40 = ">30 - 40"
40 - 50 = ">40 - 50"
50 - 60 = ">50 - 60"
60 - 70 = ">60 - 70"
70 - 80 = ">70 - 80"
80 - 90 = ">80 - 90"
90 - 99 = ">90 - 99"
99 - HIVALUE = ">99"
END
LIST Age AgeRange
```

Example 2: The RECODE command is used to generate an age range. The ELSE clause ensures that any values not captured in the recoding process are assigned a default value. In this case, any values greater than 60 are assigned "Senior."

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE AgeRange TEXTINPUT
RECODE Age TO AgeRange
LOVALUE - 0 = "<=0"
0 - 10 = ">0 - 10"
10 - 20 = ">10 - 20"
20 - 30 = ">20 - 30"
30 - 50 = ">30 - 50"
50 - 65 = ">50 - 65"
ELSE = "Senior"
END
LIST Age AgeRange
```

Example 3: The RECODE command is used to generate a detailed age range from 0 to 70 in increments of three. Note that a single RECODE command is limited to approximately 12 conditions because of query size limitations inherent in the Access database format. The desired age categories would require more than 12 recodes. To work around this problem, only 10 recodes are done at a time and are separated by a series of SELECT, WRITE, and READ commands. The first WRITE command creates a new temporary table (or overwrites an existing one) that stores only records that contain recoded values. The remaining records are not written out because of the SELECT command. Each subsequent block of recoded values is written to the same file using the APPEND parameter. By the time the code is done executing, the table T1 contains all of the recoded data.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE AgeRange TEXTINPUT
RECODE AGE TO AgeRange
LOVALUE - 0 = "<=0"
0 - 3 = ">0 - 3"
3 - 6 = ">3 - 6"
6 - 9 = ">6 - 9"9 - 12 = ">9 - 12"
12 - 15 = ">12 - 15"
15 - 18 = ">15 - 18"
18 - 21 = ">18 - 21"
21 - 24 = ">21 - 24"
24 - 27 = ">24 - 27"
END

SELECT NOT AgeRange = (.)
WRITE APPEND "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="{C:\Epi_Info_7\Projects\Sample\Sample.prj}: T1 *

READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE AgeRange TEXTINPUT
RECODE Age TO AgeRange
27 - 30 = ">27 - 30"
30 - 33 = ">30 - 33"
33 - 36 = ">33 - 36"
36 - 39 = ">36 - 39"
39 - 42 = ">39 - 42"
42 - 45 = ">42 - 45"
```

```
45 - 48 = ">45 - 48"
48 - 51 = ">48 - 51"
51 - 54 = ">51 - 54"
54 - 57 = ">54 - 57"
END

SELECT NOT AgeRange = (.)
WRITE APPEND "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="{C:\Epi_Info_7\Projects\Sample\Sample.prj}: T1 *

READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE AgeRange TEXTINPUt
RECODE Age TO AgeRange
57 - 60 = ">57 - 60"
60 - 63 = ">60 - 63"
63 - 66 = ">63 - 66"
66 - 69 = ">66 - 69"
69 - 70 = ">69 - 70"
70 - HIVALUE = ">70"
END

SELECT NOT AgeRange = (.)
WRITE APPEND "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="{C:\Epi_Info_7\Projects\Sample\Sample.prj}: T1 *
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:T1
LIST Age AgeRange
```

REGRESS

Description

This command performs multivariate linear regression with support for automatic dummy variables and multiple interactions. If a variable has more than two values, it is automatically turned into a series of 'yes/no' variables called 'dummy variables', one for each extra value.

Syntax

```
REGRESS <dependent variable> = <independent variable(s)> [NOINTERCEPT]
[OUTTABLE=<tablename>] [WEIGHTVAR=<weight variable>] [PVALUE=<PValue>]
```

- **<dependent variable>** represents the dependent variable.
- **<independent variable(s)>** is an independent variable that can be a numeric variable, a non-numeric variable, or a variable surrounded by parenthesis. Any text or yes-no variable, or a variable surrounded with parenthesis, is automatically recoded into dummy variables. A dummy variable is created for all but one of the levels of a variable. This dummy variable measures the contribution of its level relative to the excluded level. Interactions are specified by * between variables.
- **<weight variable>** represents a variable describing each data row's contribution to the regression
- **<tablename>** is the table to store the residuals. If no table name is present, no residuals are produced.
- **<PValue>** represents the size of the confidence intervals; may be specified as percent or decimal; if greater than .5, 1-PValue is used. The default is 95%.

Comments

REGRESS uses the least-squares method for determining coefficients.

Example

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:BabyBloodPressure
REGRESS SystolicBlood = AgeInDays Birthweight
```

RELATE

Description

This command links one or more tables to the current dataset during analysis, using a common identifier to find matching records. The identifier may span several fields, in which case values in each of the fields must match.

Syntax

```
RELATE <table specification> [<key(s)>] {ALL}
```

- The <table specification> represents the type and name of a data table to be read as the Related file.
- The <key(s)> represent one or more expressions that designate keys on which the match or relate will be performed. When multiple expressions are used, they are connected with AND. These following are in the form:

<ExpressionCurrentTable> :: <ExpressionRelateTable>

- The ALL represents records in the table(s) already READ or related for which there is no corresponding record in the RELATE table will be included in the data with null values for variables in the RELATE table. If absent, only records with corresponding records in the RELATE table will be included in the data.

Program Specific Features

Analysis

If the relationship was created in Form Designer, Classic Analysis can relate the two tables without need of a key expression.

Form Designer

- Grid tables and relate buttons help create parent-child relationships.
- Grid tables have a prefix of recgrid table.

Comments

To use RELATE, at least one table must have been made active with the READ command. The table to be linked must have a key field that identifies related records in the other table. In Epi Info 7, the keys in the main and related tables or files might not have to have the same name.

The expressions in the key are the names of variables in the tables that will be used to determine which records match each other. More than one key pair ("multiple keys") can be designated, separated by AND.

After issuing the RELATE command, the variables in the related table may be used as if they were part of the main table. Where variable names are duplicated in the related tables, the variable names will be suffixed with a sequence number. In referring to a variable in a related table, you may (optionally) use the form HOUSE.AGE to represent the variable AGE in the form HOUSE. This will distinguish it from another variable AGE that might be in the main table.

Frequencies, cross-tabulations, and other operations involving data in both the main and related tables can be performed. To preserve the linked structure, the WRITE command may be used to create a new table. More than one table may be related to the main table or related table by using successive RELATE commands.

Example

The records from the RHepatitis data tables are related to the Surveillance data table.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Surveillance
RELATE RHepatitis GlobalRecordId :: [FKEY]
```

ROUTEOUT

Description

This command directs output to the named file until the process is terminated by CLOSEOUT. Output from commands (e.g., FREQ and LIST) is appended to the same output file as it is produced.

Syntax

```
ROUTEOUT <filename> {REPLACE | APPEND}
```

- The <filename> represents an HTM document where the output will be stored. If no directory is specified, use the directory of the current project.
- The REPLACE | APPEND keyword controls what happens to an existing file with the same name. If REPLACE is specified, any existing file of the same name is deleted prior to writing. If APPEND is specified, new output is appended to any existing file with the same name.

Comments

Epi Info 7 sends output to an HTML document that any browser can be read. If no output is selected, Epi Info 7 creates a new file (typically called OUTXXX.HTM) where XXX is a sequential number. Output files are placed in the same directory as the current project. The prefix for output files and a starting sequence number can be changed from the Storing Output command located in the Output folder.

If no path is specified, or if the directory does not exist, the output file is created in the directory of the current project.

Example

The output generated by running the commands below is sent to the file Outbreak1.htm in the C:\Epi_Info_7\Projects\folder.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
ROUTEOUT "C:\Epi_Info_7\Projects\Outbreak1.htm" REPLACE
SET STATISTICS=COMPLETE
TABLES Vanilla Ill STRATAVAR=Sex
MEANS Chocolate Ill
CLOSEOUT
```

RUNPGM

Description

This command runs a stored Classic Analysis program. This command is similar to an INCLUDE file or subroutine in other systems.

Syntax

```
RUNPGM '<file name>':"<program>"
RUNPGM '<file name>'
```

- The <file name> represents the path and filename for the MDB or PGM file where the program is stored. If the path or filename contains a space, it must be enclosed in single quotes. If the program to be run is in the current project, the path need not be supplied.
- The <program> represents the Program name. If the program name contains a space, it must be enclosed in double quotes. This is not used for text files.

Comments

Since the filename can include any path and database name, the program to be executed can be stored in a different database.

Examples

Example 1: The program editor code contained in the Statistics.pgm file is run.

```
RUNPGM "C:\MyProject_Folder\Sample\Sample.prj":Statistics
```

SELECT

Description

This command allows an expression to be specified that must be true to process a record. It can also be called a data filter. If the current selection is Age>35, then only those records with age greater than 35 are selected. SELECT used alone without an expression cancels all previous SELECT statements. SELECT statements are cumulative until canceled. Therefore, Select Age > 34, Select Sex = “Male" will choose males over the age of 34. Cancel Select before doing Select Sex = “Female" because you will get only records that are male and female, or none at all.

Syntax

```
SELECT <expression>
```

- The <expression> represents any valid Epi Info 7 expression.

Comments

SELECT expressions are cumulative so that the two expressions:

SELECT Age > 35

SELECT Sex = "F"

are equivalent to

SELECT (Age > 35) AND (Sex = "F")

Examples

Example 1: A subset of the data contained in the Food History table is selected. In this case, only records where the patient is female and interviewed after 04/15/2011, or where the patient is male and interviewed after 06/01/2011 are selected to be in the subset. Note the use of parentheses to show relationships.

```
READ {C:\Epi_Info_7\Projects\Ecoli\EColi.prj}:FoodHistory
SELECT ((DateofInterview > 04/15/2011) AND (Sex =”F-Female")) OR ((DateofInterview >
05/15/2011) AND (Sex = "M-Male"))
LIST * GRIDTABLE
```

Example 2: A subset of the data contained in the Food History data table is selected. In this case, only records where the patient's first name is "Pam" are selected to be in the subset. Note that "Huber" is not case sensitive, and the SELECT command would have also selected "HUBER" and "huber”.

```
READ {C:\Epi_Info_7\Projects\Ecoli\EColi.prj}:FoodHistory
SELECT LastName = "Huber"
LIST * GRIDTABLE
```

Example 3: A subset of the data contained in the Food History data table is selected. In this case, only records where the patient's last name starts with the letters "Sch" and ends with the letter "r" are selected to be in the subset.

```
READ 'C:\Epi_Info\Refugee.mdb':Patient
SELECT LastName LIKE "Sch*r"
LIST LastName
```

Example 4: A subset of the data contained in the Oswego data table is selected. In this case, only records where the patient is ill are selected to be in the subset.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SELECT Ill = (+)
LIST Name Age Ill
```

Example 5: A subset of the data contained in the Oswego data table is selected. In this case, all records except those that do not have a value for the TimeSupper variable (the field was left blank during data entry) will be selected to be in the subset.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SELECT NOT TimeSupper = (.)
LIST TimeSupper DateOnset Sex Ill
```

Example 6: A subset of the data contained in the Oswego data table is selected. Two SELECT commands have a cumulative effect so that the two expressions are equivalent to SELECT (Age > 30) AND (Sex = "Male"). Only records where the age is greater than 30 and the sex is male will be selected.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SELECT Age > 30
SELECT Sex = "Male"
LIST Age Sex
```

Example 7: A subset of the data contained in the Oswego data table is selected. Two SELECT commands have a cumulative effect making the two expressions equivalent to SELECT (Age > 30) AND (Age < 20). Only records where the age is greater than 30 and less than 20 are selected. Because these two conditions are mutually exclusive, the LIST command produces no output. A CANCEL SELECT command may be issued to clear the current selection criteria.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SELECT Age > 30
SELECT Age < 20
LIST Age Sex
```

SET

Description

This command provides various options that affect the performance and output of data in Classic Analysis. These settings are utilized whenever you use the Classic Analysis program.

Syntax

```
SET [<parameter> = <value>]
```

The <parameter> = <value> represents various elements on the form. Any number of elements may be used in a single SET statement.

Parameter	Values	Response
(-)	"<Text>"	In Boolean variables, NO will be represented as <Text>.
(.)	"<Text>"	Variables with missing values will be represented as <Text>.
(+)	"<Text>"	In Boolean variables, YES will be represented as <Text>.
YN	"<Text1>"	Sets displayed text for Yes, No, and Missing to Text1, Text2, and Text3 respectively.
PROCESS	NORMAL	Deleted records are not included.
	DELETED	Only deleted records are included.
	BOTH	Deleted records are included.
DELETED	YES or (+)	Deleted records are included.
	NO or (-)	Deleted records are not

Parameter	Values	Response
		included.
	ONLY	Only deleted records are included.
MISSING	(+) or ON	Include missing values for analysis.
	(-) or OFF	Do not include missing values for analysis.

Program Specific Feature

In the command generator, selecting Save All generates a SET command, which contains all current settings. Selecting Save Only or OK generates a SET command, which contains only changed settings. To force a SET command to be generated for a current value of a setting, change its value and change it back again.

Example

```
SET MISSING=(-)
```

SORT

Description

This command allows a sequence to be specified for records to appear in LIST, GRAPH, and WRITE commands. If no variable names are specified after the SORT command, the current sort is cleared, and subsequent record outputs will be in the order of the original data table. If one variable name is given, records are sorted using that variable as the "key". If more than one variable is specified, records will be put in order by the first variable, then within a group with the same value of variable 1, the ordering will be by variable 2, etc.

Syntax

```
SORT <variable> {DESCENDING}
```

- <variable> represents the variable to be sorted by.
- DESCENDING indicates that the sort order is descending; if not specified, ascending order will be used.

Comments

The parameter DESCENDING must be placed next to the variable to be sorted in descending order. Should several variables be sorted in descending order, one DESCENDING should be included for each of them.

Examples

Example 1: The data is sorted by Age in ascending order.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SORT Age
LIST Age Sex Ill GRIDTABLE
```

Example 2: The data is sorted by Age in descending order. If two or more records have the same value for Age, the records are sorted by Ill in descending order. If two or more records have the same value for Ill, the records are sorted by Sex in descending order.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
SORT Age DESCENDING Ill DESCENDING Sex DESCENDING
LIST Age Sex Ill GRIDTABLE
```

SUMMARIZE

Description

This command creates a new table containing summary statistics for the current dataset or its strata.

Syntax

```
SUMMARIZE varname::aggregate(variable) [varname::aggregate(variable) ...] TO tablename
STRATAVAR=variable list {WEIGHTVAR=variable}
```

- Available aggregates are COUNT, MIN, MAX, SUM, FIRST, LAST, AVG, VARiance and STandardDEViation (Sum, Avg, Var and StDev available only for numeric fields). COUNT may be used without a variable in parenthesis to indicate that a count of the number of records in the table or strata is desired. You can also use COUNT with a variable in parenthesis to indicate that the number of records in the table or strata with non-missing values of the specified group is desired. FIRST and LAST are based on the current sort order.

Comments

Classic Analysis creates a new table or appends to an existing table (tablename) containing variables (varname) which represent aggregates of variables in the current data source (aggregate[variable]). The aggregates are computed for each group of records, determined by the STRATAVARs, which are also included in the table. Available aggregates are COUNT, MIN, MAX, SUM, AVG, VARiance and STandardDEViation (Sum, Avg, Var and StDev available only for numeric fields). COUNT may be used without a variable in parenthesis to indicate that a count of the number of records in the group is desired, or with a variable in parenthesis to indicate that the number of records in the group with non-missing values of the specified group is desired.

This command solves some recurring problems for programmers. One is computing percents; it is difficult to get a denominator. Another is determining the earliest or latest date in a list of relevant dates, or the highest or lowest of a series of measurements. Many problems can be solved with the OUTTABLE from a TABLES or FREQ command, or with self-joins, but this provides a straightforward method to achieve these results.

Note: Multi-line (memo) fields are not permitted.

Example

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:EvansCounty
SUMMARIZE Average_Age :: Avg(AGE) Average_DBP :: Avg(DBP) Number_Records :: Count(AGE)
Std_Age :: StDev(AGE) Std_DBP :: StDev(DBP) TO SUMMARY_TABLE
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Summary_Table
LIST *
```

TABLES

Description

This command cross-tabulates specified variables in two or three dimensions. Values of the first variable appear across the top of the table while those of the second variable appear in the left margin of the table. Unique values of additional variables are represented as strata. Normally, cells contain counts of records matching the values in the corresponding marginal labels. If a WEIGHTVAR parameter is given, the cells represent sums of the weight variable. TABLES DISEASE COUNTY WEIGHTVAR=COUNT provide the same results as SUMTABLES COUNT DISEASE COUNTY in Epi6.

Syntax

```
TABLES <exposure> <outcome> {STRATAVAR=[<variable(s)>]} {WEIGHTVAR=<variable>}
{PSUVAR=<variable>} {OUTTABLE=<table>}
```

- **<exposure>** represents the variable in the database to be considered the risk factor (or * for all variables).
- **<outcome>** represents the variable in the database considered disease of consequence (or * for all variables).
- **<variable>** represents the variable in the database.
- **<table>** represents a valid table name to be used to store output.

Comments

For every possible combination of values of the strata variables, a separate table (stratum) for variable 1 by variable 2 is produced. TABLES BAKEDHAM ILL STRATAVAR=SEX produces a table of BAKEDHAM by ILL for each value of sex-one for M and one for F. TABLES BAKEDHAM ILL STRATAVAR=SEX RACE produces a separate table of BAKEDHAM by ILL for each combination of SEX and RACE-female/black, female/white, male/black, male/white, etc.

If * is given instead of a variable name, each variable in the dataset is substituted for * in turn. To analyze each variable by illness status, use the command TABLES * ILL which produces tables of SEX by ILL, AGE by ILL, etc.

It is important to consider using * or requesting multidimensional tables if the dataset is large (thousands of records), since it may produce more tables than needed in terms of time, paper, and other costs. Press **Ctrl-Break** to exit from a lengthy procedure.

For 2x2 tables, the command produces odds and risk ratios. For these values to have their accepted epidemiological meanings, the value representing presence of the exposure and outcome conditions must appear in the first row and column of the table. Epi Info yes/no variables are automatically sorted properly. The STATISTICS setting controls the detail to

which the statistics are reported. For tables other than 2x2, Chi-square statistics are computed. If an expected value is < 5, the message Chi-square not valid appears. The mid-p and Fisher exact are preferred, especially in this case.

Multiline (memo) variables cannot be used in tables. To use a Multiline variable, define a new variable and assign to it the value SUBSTRING(<old variable>,1,255) and use it in the table.

Examples

Example 1: A 2x2 table is generated showing coronary heart disease (CHD) by Catecholamine Level (CAT).

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:EvansCounty
TABLES CAT CHD
```

Example 2: A 2x2 table is generated showing coronary heart disease (CHD) by Catecholamine Level (CAT), stratified by an age group variable of type yes/no.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:EvansCounty
TABLES CAT CHD STRATAVAR=AgeG1
```

Example 3: A 2x2 table is generated for every variable in the database using Ill as the outcome variable for table.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
TABLES * Ill
```

Example 4: A 2x2 table is generated and saved to a separate table in the Sample database using the OUTTABLE parameter.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
TABLES Vanilla Ill OUTTABLE = T1
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:T1
LIST * GRIDTABLE
```

Example 5: A 2x2 table is generated showing obesity and disease outcome. The analysis is weighted by the value contained in the COUNT column.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Lasum
TABLES OB OUTCOME WEIGHTVAR=COUNT
```

Example 6: A complex sample table is generated using a stratified cluster survey.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Epi10
TABLES Prenatal VAC STRATAVAR=Location WEIGHTVAR=POPW PSUVAR=Cluster
```

TYPEOUT

Description

This command inserts text, a string or file contents, into the output. Typical uses might include comments or boilerplate.

Syntax

```
TYPEOUT "text" ([<font property>]) {TEXTFONT <color> <size>}
```

- **<text>** represents the text to be displayed.
- **<font property>** represents the properties Underline, Bold, or Italic separated by commas.
- **<color>** represents Aqua, Lime, Red, Black, Silver, Maroon, Blue, Navy, Teal, Fuchsia, Olive, White, Grey, Purple, Yellow, or Green. Color can also be represented by hexadecimal digits ###### with pairs of digits representing the amount of red, green, and blue on a scale of 0–255.

Comments

TYPEOUT with a text string is similar to TITLE except that TYPEOUT places the text once when the command is encountered. HEADER appears at the top of each segment of output until cleared.

If no text in quotation marks follows the TYPEOUT command, TYPEOUT sends the file contents specified to the current output. It can be in text or HTML.

Examples

Example 2: Displays the word "Confidential" underlined and in bold.

```
TYPEOUT "Confidential" (BOLD,UNDERLINE)
```

UNDEFINE

Description

This command removes a defined variable and any assigned values from the system.

Syntax

```
UNDEFINE <variable>
```

- The <variable> represents a defined variable.

Program Specific Feature

Permanent variables cannot be undefined from the Undefine dialog box. To undefine a permanent variable, type the syntax into the Program Editor and run the command.

Comments

A variable that already exists in the database cannot be undefined. To remove a database variable from the database, use the WRITE command with the EXCEPT modifier.

Example

```
UNDEFINE NewVar
```

UNDELETE

Description

This command will mark logically deleted records as normal.

Syntax

```
UNDELETE *
UNDELETE expression
```

Comments

Undelete * or expression causes all logically deleted records in the current selection matching the expression to be set to normal status. This applies only to Epi Info 7 forms and may not be used when using related tables.

Example

```
UNDELETE AgeInDays > 5
```

WRITE

Description

The WRITE command sends records to an output table or file in the specified format. Specifications include which variables are written, variable order, and the type of file to be written.

Syntax

```
WRITE <METHOD> {<output type>} {<project>:}table {[<variable(s)>]}
WRITE <METHOD> {<output type>} {<project>:}table * EXCEPT {[<variable(s)>]}
```

- **<METHOD>** represents either REPLACE or APPEND
- **<project>** represents the path and filename of the output.
- **<variable(s)>** represents one or more variable names.
- **<output type>** represents the following allowable outputs:

Database Type	Specifier	Element
Epi Info 7	"Epi Info 7"	<path:<table>
MS Access 97-2003	MS Access 97-2003	<path>
MS Access 2007	MS Access 2007	<path>
Excel 97-2003	MS Access 97-2003	<path>
Excel 2007	MS Access 2007	<path>
SQL Server		Server Name & Database Name
Text (Delimited)	"Text"	<path>

Comments

Records deleted in Enter or selected in Classic Analysis are handled as in other Analysis commands. Defined variables may be written to allow you to create a new Epi Info 7 file to make permanent changes. Unless explicitly specified, global and permanent variables will not be written.

To write only selected variables, the word EXCEPT may be inserted to indicate all variables except those following EXCEPT.

If the output file specified does not exist, the WRITE command will attempt to create it.

Either APPEND or REPLACE must be specified to indicate that an existing file/table by the same name will be erased or records will be appended to the existing file/table. If not all of the fields being written match those in an existing file during an APPEND, the unmatched fields are added to the output table.

Examples

Example 1: The Oswego data table (75 records) from Sample.PRJ is written to a database called SampleOutput. The destination table name is called Oswego_1. The second READ command reads the newly-created data table.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
WRITE REPLACE "Epi 2000" 'C:\Epi_Info\SampleOutput.mdb':Oswego_1 *
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego_1
```

Example 2: The Oswego data table (75 records) from Sample.PRJ is written to a database called SampleOutput three times. The APPEND method ensures that each WRITE command appends the entire data set several times. After all three WRITE commands have been run, the Oswego_2 table inside SampleOutput.mdb will contain 225 records.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
WRITE APPEND "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="C:\Epi_Info_7\Projects\SampleOutput.mdb"} : OSWEGO_2 *
WRITE APPEND "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="C:\Epi_Info_7\Projects\SampleOutput.mdb"} : OSWEGO_2 *
WRITE APPEND "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="C:\Epi_Info_7\Projects\SampleOutput.mdb"} : OSWEGO_2 *
READ {C:\Epi_Info_7\Projects\SampleOutput.mdb”}:OSWEGO_2
```

Example 3: This example shows how to make defined variables into permanent database variables. The Oswego data table from Sample.PRJ is written to a database called SampleOutput. Notice that the defined variable IncubationTime does not exist in Sample.PRJ, but after the WRITE command has executed, it is now a part of the newly-created data table Oswego_3.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE IncubationTime NUMERIC
ASSIGN IncubationTime = HOURS(TimeSupper, DateOnset)
LIST IncubationTime
WRITE REPLACE "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="C:\Epi_Info_7\Projects\SampleOutput.mdb"} : OSWEGO_3 *
READ {C:\Epi_Info_7\Projects\SampleOutput.mdb”}:OSWEGO_3

LIST * GRIDTABLE
```

Example 4: The records from the Oswego Form in the Sample database are exported to an Excel spreadsheet. By specifying age, sex, and incubation time in the WRITE command,

only those variables will be exported. This example may not work if Microsoft Excel is not installed on the computer.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
DEFINE IncubationTime
ASSIGN IncubationTime = HOURS(TimeSupper, DateOnset)
WRITE REPLACE "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="C:\Epi_Info_7\Projects\Oswego.xls";Extended Properties="Excel
8.0;HDR=Yes;IMEX=1"} : OSWEGO_1 Age Sex IncubationTime*
READ {C:\Epi_Info_7\Projects\Oswego.xls}:[OSWEGO_1$]
LIST * GRIDTABLE
```

Example 5: The records from the Oswego Form in the Sample database are exported to a text file.

```
READ {C:\Epi_Info_7\Projects\Sample\Sample.prj}:Oswego
WRITE REPLACE "Epi7" {Provider=Microsoft.Jet.OLEDB.4.0;Data Source="C:\
My_Project_Folder ";Extended Properties="text;HDR=Yes;FMT=Delimited"} : [OSWEGO#txt] *
```