

8/11/2005

MODLER for Windows
Statistical Information and Modeling System

Supplementary Programs For Data Base Management and Related Uses

MODLER Information Technologies Press
Philadelphia and Cambridge

Information in this document is subject to change without notice and does not represent a commitment on the part of the publisher nor the manufacturer. The software this manual describes is furnished under a license agreement by explicit contract and any use other than on the basis of a written contract between the original vendor and the purchaser prima facia constitutes an infringement of the copyright. The manual and the software each may be used or copied only in accordance with the terms of that contractual agreement. It is against the law to copy this software or manual onto cassette tape, disk, CDROM, or any other medium for any purpose other than the purchaser's private use, or the personal use of the purchaser's employees.

© Copyright 2002-2005 Alphametrics Corporation. All Rights Reserved

All rights reserved. No part of this publication can be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright owner. This copyright covers not only the presentation of information in this manual, but also the program's human interface, MODLER command syntax and the way in which the commands are ordered to form the language as a whole.

LIMITED WARRANTY

Neither the manufacturer nor the distributors of the MODLER software shall have any liability or responsibility to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product. This product will be exchanged within twelve months from the date of purchase if it is found to be defective in manufacture, labeling, or packaging; but except for such replacement the license of this software is without warranty or liability.

The above is a limited warranty and the only warranty made by the manufacturer, publisher, or distributors of the MODLER software. Any and all warranties for merchantability or fitness for a particular purpose are hereby excluded.

Trademarks Acknowledged

MODLER, MODLER BLUE, MODLER MBA, DATAVIEW and their derivatives are trademarks of C.G. Renfro & Associates. LOTUS 1-2-3 and WordPro are trademarks of Lotus Development Corporation. Quattro Pro and WordPerfect are trademarks of Corel Systems. Excel, MS-DOS, and Word are trademarks of Microsoft Corporation. All other product names in this publication are trademarks or registered trademarks of their respective owners.

Technical Support

Europe

Alphametrics Limited
Heath House
Princes Mews
Royston, Herts SG8 9RT
United Kingdom

Tel: +44 (0) 1763 242 277
Fax: +44 (0) 1763 243 988
Email: Support@modler.com

North America and Elsewhere

Alphametrics Corporation
PO Box 2566
Bala Cynwyd, PA 19004-6566
USA

Tel: +1 (215) 252-9271
Fax: +1 (215) 252-9279
Email: Support@modler.com

Introduction

This publication describes the data management programs that are found in the DATAMAN.EXE self-extracting zipped file that is downloadable from the www.modler.com website. These programs are DOS and Windows command line programs that are essentially designed to be used with MODLER data banks in a supplementary role to the MODLER, MODLER BLUE, and DATAVIEW programs. They are easy to use and can be employed from Windows applications either in the context of a DOS window or a macro command file, although certain of them are perhaps best used interactively.

From within MODLER, MODLER BLUE, or DATAVIEW you can launch a DOS session simply by issuing the command:

COMMAND

on its own. Whether you are using the DOS or Windows versions of these programs, you will immediately be in DOS mode. You can return to non-DOS mode by issuing the command:

EXIT

If you are using the Windows versions of these programs, you can alternatively click on the menu item **File** and choose the option **Execute Another Program**. Then, once you state the name of the program, you will be executing in DOS mode; however, upon completion of the task, you will in this case immediately (and essentially automatically) be returned to standard Windows mode.

Considering initially the four programs that are the most straightforward in application, ANALYZE, BCHECK, PROBE and CHGNA, these are run by stating the program name followed by the name of a bank. Thus, for example:

```
ANALYZE USQBANK
BCHECK USQBANK
PROBE USQBANK
CHGNA USQBANK
```

Where in each case the bank name (USQBANK) is presumed to have the extent .BNK as a default.

The ANALYZE program evaluates the named bank, checking it for a variety of possible errors. If any are discovered, a separate file bankname.log is created

immediately that describes them, where “bankname” is the name of the bank being validated. In the present case, this file would consequently be given the name USQBANK.LOG. Even in the absence of errors, an abbreviated statement of the bank’s primary characteristics is generated. ANALYZE is an updated version of BCHECK that among other things recognizes accented text characters in French, Spanish, and other non-English languages.

The BCHECK program also checks the named bank for errors. If any are discovered, a separate file bankname.LOG is created describing them. As in the above case, it would of course be given the name USQBANK.LOG. In the absence of errors, an abbreviated statement of the bank’s primary characteristics is generated. The BCHECK program should *not* be used for data banks documented in languages other than English and it should not be used for data banks created using the windows versions of MODLER, MODLER BLUE and DATAVIEW, or DOS versions later than version 5.10, as will be described.

The PROBE program is an extended version of ANALYZE. It evaluates the named bank, checking it for a variety of possible errors. If any are discovered, a separate file bankname.log is created immediately that describes them, where “bankname” is the name of the bank being validated. In the present case, this file would consequently be given the name USQBANK.LOG. Even in the absence of errors, an abbreviated statement of the bank’s primary characteristics is generated. Similarly to ANALYZE, PROBE is an updated version of BCHECK that among other things recognizes accented text characters in French, Spanish, and other non-English languages. However, the most significant characteristic of PROBE is that if errors are discovered, the details provided are much more extensive than for either ANALYZE or BCHECK; it should therefore generally be used in addition to either of these other programs, once any errors are discovered.

To understand the use of the special purpose CHGNA program requires a little more explanation. In the context of data base management, MODLER, MODLER BLUE, and DATAVIEW each allow for the possibility that a data series may be missing observation values for any of a variety of reasons. Such missing values are assigned a specific numeric value, which is today the value $-1E11$ (-1×10^{11}), a number that would never be encountered as an actual observation value for a series. However, originally, in 1969, the corresponding NA value assigned was -9999.99 , a value that was seldom if ever found as an actual macroeconomic data value in those days, but has since become relatively common; for example, in the case of trade or government deficit values. The CHGNA program reads through a data bank series by series and converts the value used to indicate a not available value from -9999.99 to $-1 E 11$. Versions of MODLER, MODLER BLUE and DATAVIEW prior to 5.20 used the old value for NA. All versions since 5.20, inclusive, create banks with the new NA value, which includes all Windows versions of these programs, since the first Windows version of MODLER post-dates DOS version 5.20.

The program UNCHGNA is also run by stating the program name followed by the name of a bank:

UNCHGNA USQBANK

and is best understood as the opposite of CHGNA. UNCHGNA converts a new type bank into an old type of bank. It is included simply to allow specific users to reverse the conversion process for NA observation values, changing this value from -1.0E11 to the value -9999.99. Although version 5.20 of MODLER, MODLER BLUE, and DATAVIEW was released during the early 1990s, these programs are sufficiently stable and beloved of long-term MODLER users that there are still pre-5.20 copies in use today, notwithstanding that such versions therefore pre-date the release of Windows 3.1, to say nothing of Windows 95, 98, NT, Windows 2000 or Windows XP, yet can be and have been used with all these Windows versions.

Note also that, in all cases, the bankname's extent can be omitted; that is, so long the default extent .BNK is used for a data bank, which is strongly recommended. In addition, you should always bear in mind that making **any** changes to existing banks is inherently risky. Therefore, *please be sure in all cases to back up your banks and other pertinent data files before using any of the programs described in this document.*

Finally, the two programs:

CDOC.EXE
SLOAD.EXE

respectively allow you to manage globally the documentation of data bank series and to create or update data banks using data copied from a MODLER solution file. These programs offer a variety of options while performing these tasks and are described in some detail .

ANALYZE.EXE

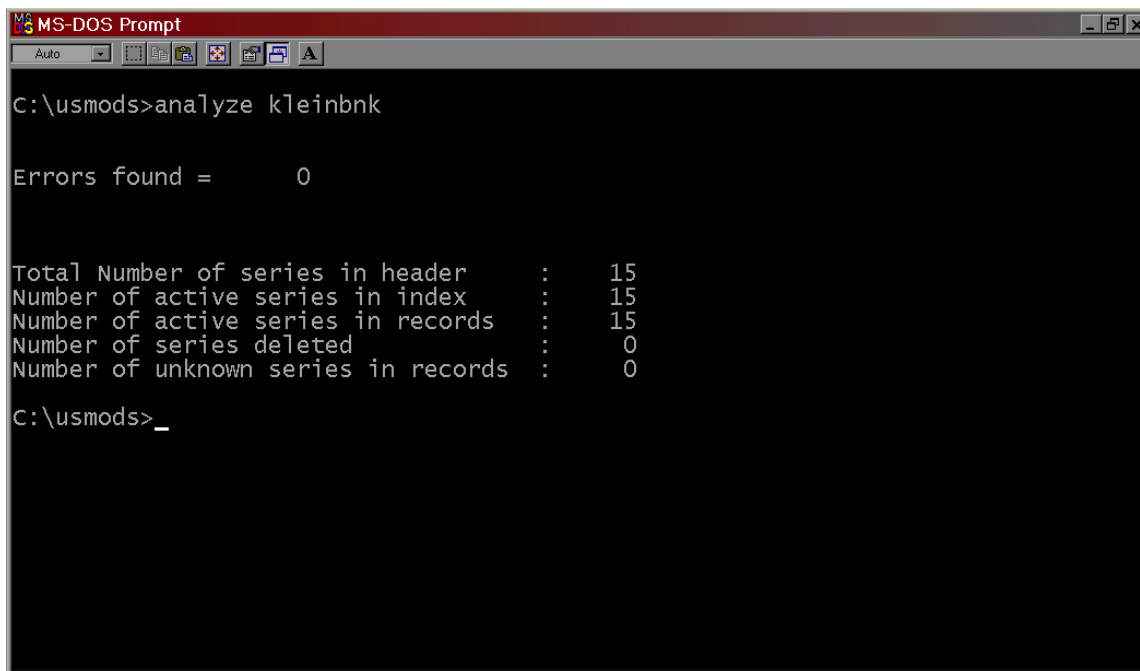
Data Bank Validation Program

The ANALYZE program evaluates a data bank, checking to see if it has any compositional errors, such as data series that are not properly indexed or that otherwise exhibit some pathology. The program is run using the syntax:

```
ANALYZE bankname
```

where bankname is the name of a MODLER, MODLER BLUE, or DATAVIEW data bank. Provided that the bankname has the default extent .BNK, this extent can be omitted.

Figure 1 provides an example of the summary output of this program when no data bank errors are found. Note that this display states the number of series in the bank according to the data bank header, the index, and data series records respectively. One of the ways in which a data bank can become corrupted is to loose all or part of its index, or somehow for the number of series as recorded in the header to cease to match the number of series records.



```
MS-DOS Prompt
Auto
C:\usmods>analyze kleinbnk

Errors found =      0

Total Number of series in header      :      15
Number of active series in index      :      15
Number of active series in records    :      15
Number of series deleted               :       0
Number of unknown series in records   :       0

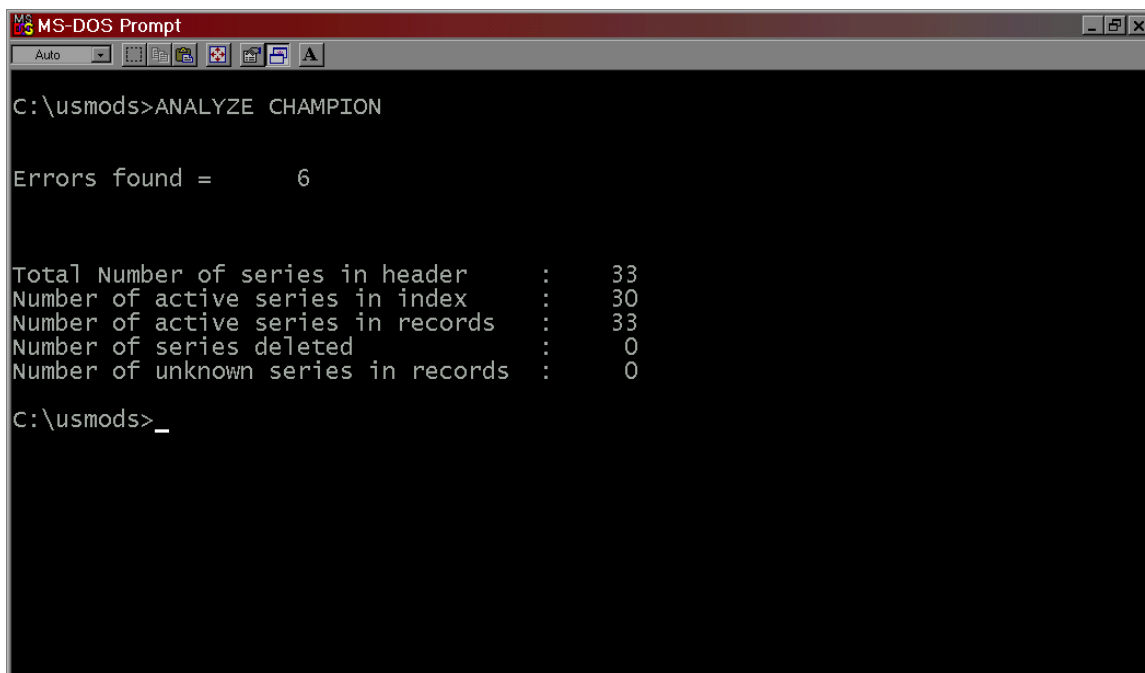
C:\usmods>_
```

Figure 1. Analyze program summary output

The information given here is not all error-related. For instance, this bank happens not to contain any deleted series. But by design if a series is created and then deleted, it

remains in a data bank. The deletion operation involves simply turning the series “off.” If needed, such a series can later be restored to use. Deleted series are therefore not, per se, an indication of a corrupted data bank, but they do count in terms of the number of records used. The total number of data bank records available consist of the sum of active series records, deleted series records, and unknown series records, the latter consisting of as yet unused records which therefore are not recognized as valid series.

In contrast to Figure 1, Figure 2 illustrates an instance of a bank that contains errors, in this case correctable. Note that the number of active series in the index is less than the number of active series records. One of the implications of this type of data bank error is that certain of the series ostensibly in the data bank cannot be used. A series is retrieved for use by name and in order to do this that series name must be present in the data bank index. The question this circumstance raises is how to determine what the specific problem is? It would be useful to know, for example, which series appear in the records that do not appear in the index. Ideally, it would also be nice to know why the problem has occurred, although this is generally very difficult to determine after the fact in isolated cases.



```

MS-DOS Prompt
Auto
C:\usmods>ANALYZE CHAMPION

Errors found =      6

Total Number of series in header      :    33
Number of active series in index      :    30
Number of active series in records    :    33
Number of series deleted              :     0
Number of unknown series in records   :     0

C:\usmods>_

```

Figure 2. An Example of a Corrupted Data Bank

This second level of error analysis is performed using the error log file. Whenever the ANALYZE program discovers data bank errors, it automatically creates a log file, called bankname.LOG, where “bankname” is the name of the bank being analyzed, in this case CHAMPION.BNK. This log provides a detailed description of the errors and can be viewed using almost any text editing or word processing program. In the Windows context, Notepad and/or WordPad are immediately available to use for this

purpose. Figure 3 displays a portion of the file CHAMPION.LOG in the context of Wordpad.

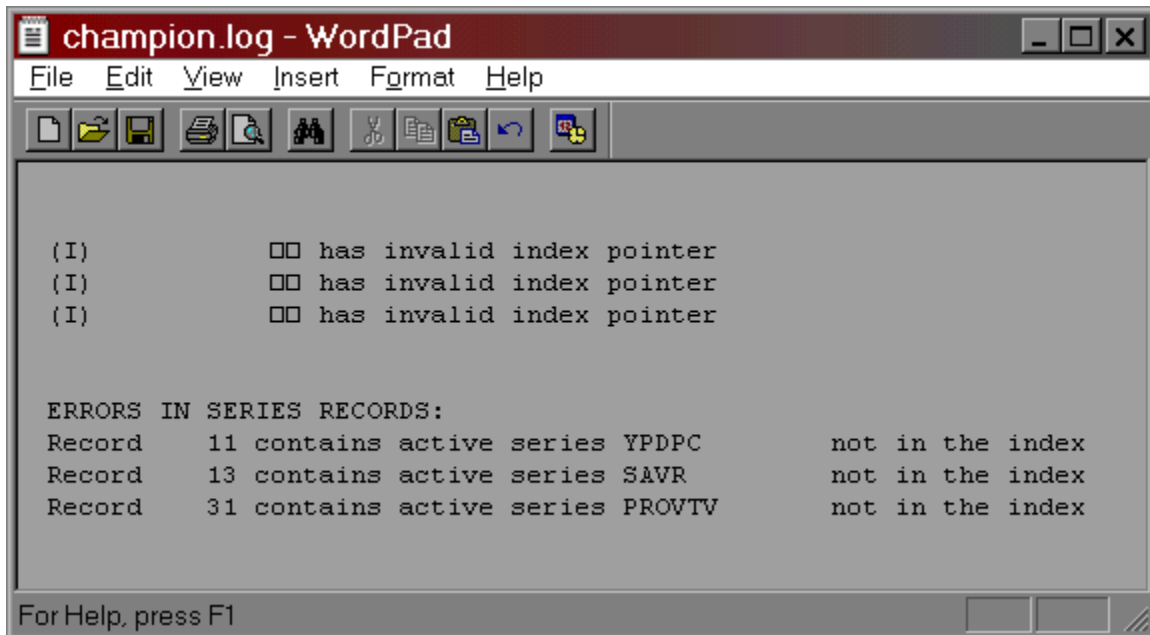


Figure 3. CHAMPION.LOG Error File

The origin of the problem that Figure 3 illustrates is difficult to determine ex post, but what has occurred is that the index has three elements that each have an invalid index pointer. Actually, the fact that no series name appears in each of these cases implies that the problem is really that the index elements are effectively empty. Therefore it is almost irrelevant that the index pointers do not point at some existing series. The symbols you see instead of series names are extraneous ASCII box characters, rather than the alphabetic characters that might be expected. Evidently, the series names were somehow overwritten by “garbage.”

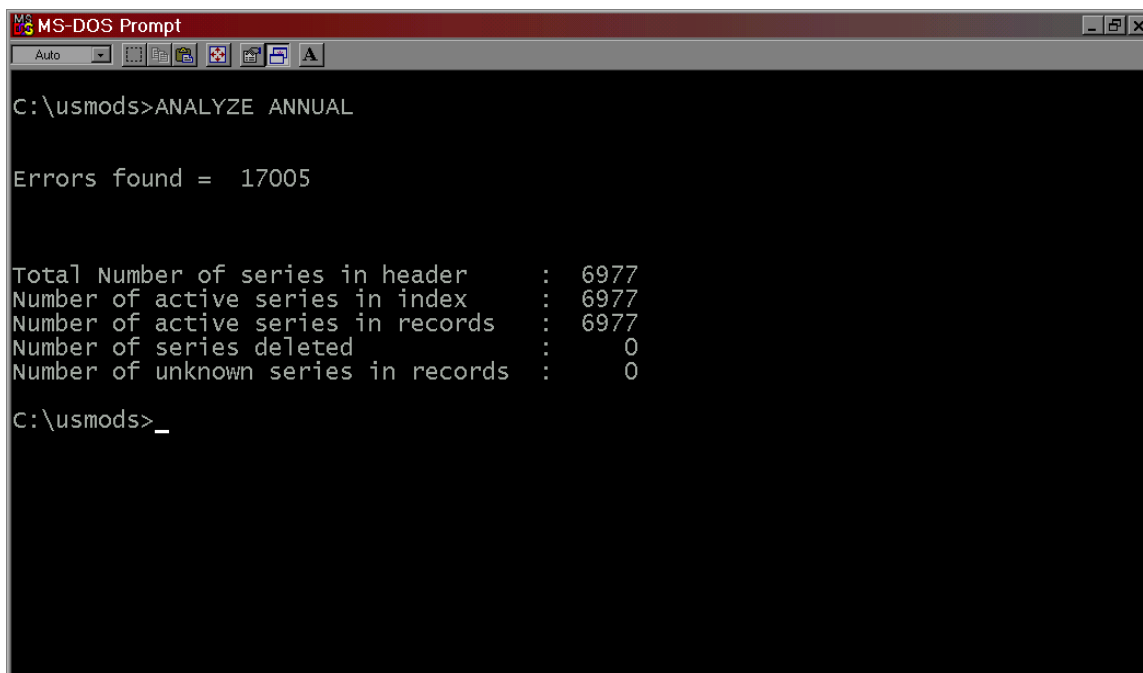
Further information is provided by the text that follows: ERRORS IN SERIES RECORDS. Ostensibly, records 11, 13, and 31 contain valid data series and these are apparently the ones that are missing from the data bank index. In fact, the inference that can be drawn from Figure 3 is that CHAMPION.BNK might be reconstructable: the series records all appear to be complete. The problem seems to be limited to the bank index.

The following can be done: first, using MODLER, MODLER BLUE, or DATAVIEW, create a brand new data bank. Call it (for example) TEMP.BNK. Next, close all banks. Then re-open TEMP.BNK as a **STORAGE** bank and CHAMPION.BNK as an **ACCESS** bank. Once you have done this, issue the command:

```
COPY SERIES=1-33 FROM CHAMPION TO TEMP
```

The logic here is that we know, from Figure 2, that the number of active series in CHAMPION.BNK is 33. Furthermore, we know that the index to this bank is corrupted, so that we do not want to use that index in any way. Thus the choice of 1-33, which tells MODLER (or MODLER BLUE or DATAVIEW) to copy the first 33 series records from CHAMPION.BNK. During the copy process, as a matter of course, a new index will be built for the bank TEMP.BNK. The net result should be the creation of a bank, TEMP.BNK, that both contains the 33 series and has a good index. After you have saved a backup of CHAMPION.BNK, you can then rename TEMP.BNK to CHAMPION.BNK. Finally, run ANALYZE.EXE again to verify that the new CHAMPION bank is error-free.

Another example of a bank that contains errors is illustrated by Figure 4. These errors are not, however, evident in the number of series in the header, index, or records, but instead are more specific to the individual series records. We infer this from the fact that the index, the number of active series, and the number of records all match. At the same time, there are still apparently 17005 errors in the bank.



```

MS-DOS Prompt
Auto
C:\usmods>ANALYZE ANNUAL

Errors found = 17005

Total Number of series in header      : 6977
Number of active series in index      : 6977
Number of active series in records    : 6977
Number of series deleted               : 0
Number of unknown series in records   : 0

C:\usmods>_

```

Figure 4. Analysis of ANNUAL.BNK

Figure 5 displays only a small portion of ANNUAL.LOG. However, this is enough to tell us the general nature of the problem. From the top working down, note first at the left the bracketed number (00001), which refers to the series record number 1. Each of the lines that begin (00001) refer to this series. Seemingly, it is named AA and has something to do with “Auto Output” which is stated in Billions of \$, as implied by the bracketed (Bil.\$). The units code is US\$, obviously for US dollars. The source code is BEA, for the Bureau of Economic Analysis. However, there are a lot of extraneous boxlike characters, but the real give away is the notation in each case the element

“contains control characters.” What has occurred is that whoever originally created the bank, which was probably done using a huge macro file, created the macro file with a text or word processing package that did not produce simple ASCII text.

```

(H) Bank has invalid type:      0
(00001) AA                    description contains control characters:
      Auto Output (Bil.$)
(00001) AA                    units contains control characters: US$
(00001) AA                    source contains control characters: BEA
(00002) AANUA                 description contains control characters:
      Ward's Auto Assemblies (NSA, Thous.Units)
(00002) AANUA                 units contains control characters: US$
(00002) AANUA                 source contains control characters: BEA
(00003) AB                    description contains control characters:
      Total Bankruptcy Filings, U.S. (Units)
(00003) AB                    source contains control characters: US$
(00004) ABCSUA                description contains control characters:
      Auto Imports from Canada, BEA (SA, Thous.Units)
(00004) ABCSUA                units contains control characters: US$
(00004) ABCSUA                source contains control characters: BEA
(00005) ABMSUA                description contains control characters:
      Auto Imports from Mexico, BEA (SA, Thous.Units)
(00005) ABMSUA                units contains control characters: US$
(00005) ABMSUA                source contains control characters: BEA
(00006) ABPRSUA               description contains control characters:
      Auto Production, BEA (SA, Thous.Units)
(00006) ABPRSUA               units contains control characters: US$
(00006) ABPRSUA               source contains control characters: BEA
(00007) ABPSUA                description contains control characters:
      BEA Auto Production Estimate (SA, Mil.Units)
(00007) ABPSUA                units contains control characters: US$
(00007) ABPSUA                source contains control characters: BEA
(00008) ABXSUA                description contains control characters:
      Exports of Autos, BEA (SA, Thous.Units)
(00008) ABXSUA                units contains control characters: US$
(00008) ABXSUA                source contains control characters: BEA
(00009) ACA                   description contains control characters:
      Auto Output (Bil.Fxd.96$)
(00009) ACA                   units contains control characters: US$
(00009) ACA                   source contains control characters: BEA
(00010) ACNA                  description contains control characters:
      PCE: New Autos (Bil.$)
(00010) ACNA                  units contains control characters: US$
(00010) ACNA                  source contains control characters: BEA
(00011) ACNCA                 description contains control characters:
      Autos, Real New: Personal Consumption Expenditures (SA, Bil.Fxd.1992$)
(00011) ACNCA                 units contains control characters: US$
(00011) ACNCA                 source contains control characters: BEA
(00012) ACNDHUA               description contains control characters:
      PCE: New Domestic Autos (SA, Mil.Chn.1996$)
(00012) ACNDHUA               units contains control characters: US$
For Help, press F1

```

Figure 5. ANNUAL.LOG

Control characters, as a phenomenon, consist of visually unrepresentable ASCII character codes. codes for tab spaces, line feeds, carriage returns, and other such things that instruct printers, CRTs and other devices, telling them how to display text. However, these codes, if they get copied into a data bank, generally gum up the works.

At minimum, the problem is that what may appear to be a single space (the equivalent of a single blank or other character on the screen) might be a tab character, which can cause printed output to lurch crazily across a page. Or the character can be a “bell” character, which when encountered displaying or reading the file, causes the computer’s bell to sound. Alternatively, the appearance of such characters can cause series names to become unrecognizable: the computer does not necessarily interpret AA followed by a blank as being the same thing as AA followed by a bell character or a tab character.

ANNUAL.BNK is actually a marginally usable bank. For example, if you look at Figure 6, you will see that it displays the series AA. Note that the description, source, and units of this series are properly displayed, and that generally nothing seems to be amiss. The problem occurs in other contexts. For instance, attempts to display portions of the data bank index result in error messages to the effect that there are “Too many series names found in the search interval.” Particularly when banks, such as this one, grow to 6000 or more series in size it is important to be able to deal with series in bulk.

Annual Data						
	I	II	III	IV	V	VI
1947-1952	7,2000	8,8000	11,900	15,400	13,300	12,000
1953-1958	16,100	14,700	21,200	16,900	19,400	14,400
1959-1964	19,400	21,400	17,700	22,400	25,100	25,900
1965-1970	31,300	30,300	27,700	35,200	34,900	28,800
1971-1976	39,200	41,200	46,300	39,800	40,700	55,600
1977-1982	65,200	69,700	68,300	60,500	70,600	65,700
1983-1988	88,300	103,800	114,200	118,800	115,200	121,600
1989-1994	121,800	113,100	102,300	111,700	121,900	133,300
1995-2000	130,500	126,100	126,700	127,300	126,100	

Figure 6. Print Out of Series AA from ANNUAL.BNK

As the foregoing discussion illustrates, the ANALYZE program has been created in order to identify data bank errors. Sometimes, as when the individual series records are complete, these errors can be corrected. Once you have identified that a bank contains compositional errors you can contact your technical support representative, who may be able to tell you immediately how to solve the problem. In other instances, the errors may take more effort to correct.

The ANALYZE program is designed to be quite sensitive to potential data bank problems and can in some cases indicate that a bank contains errors even when the bank is, to all intents and purposes, perfectly usable. There are certain data bank “errors” that are cosmetic. This is only to say that there are some rules of data bank construction that can be broken without affecting the use of a bank. However, there are no reported cases that ANALYZE has found a bank to be error free that the bank then has not worked properly.

BCHECK.EXE

Data Bank Validation Program

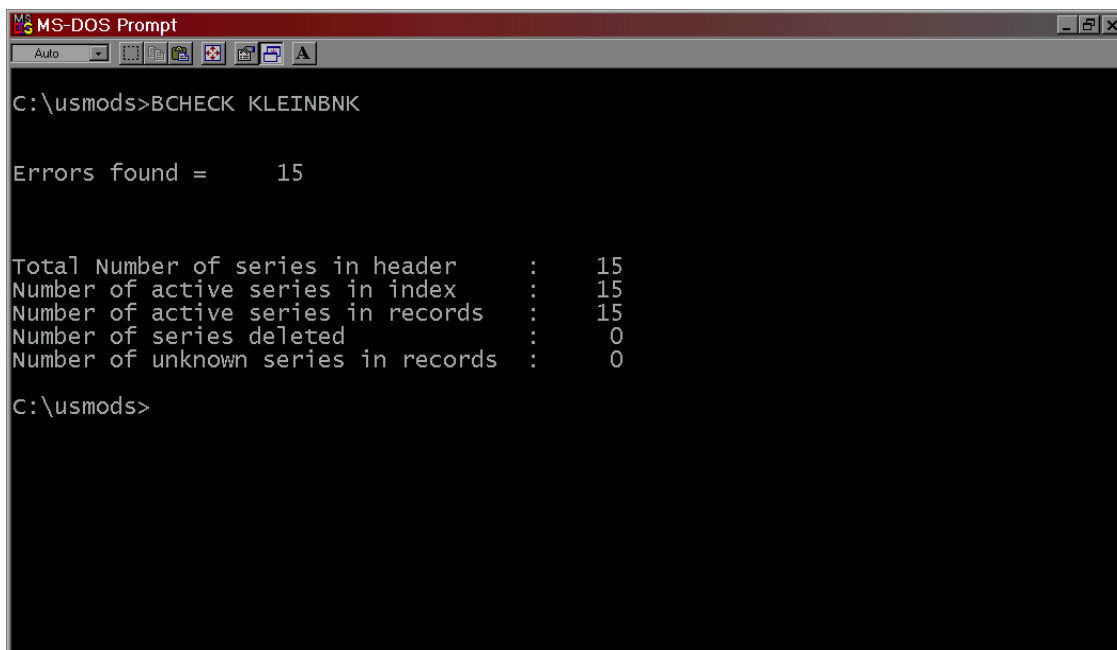
Like ANALYZE and PROBE, BCHECK is validation program that evaluates a data bank, checking to see if it has any compositional errors, such as data series that are not properly indexed or that otherwise exhibit some pathology. The execution syntax is:

```
BCHECK bankname
```

where bankname is the name of a MODLER, MODLER BLUE, or DATAVIEW data bank. Provided that the bankname has the default extent .BNK, this extent can be omitted.

Compared to ANALYZE and PROBE, BCHECK is both older, specifically designed to be used with data banks compatible with MODLER and DATAVIEW versions earlier than 5.20, and strictly for use with English language data banks. In contrast, ANALYZE and PROBE can recognize accented text characters in French, Spanish and other non-English languages.

Figure 7 provides an example of the summary output of this program when no data bank errors are found. Note that this display reveals the number of series in the bank according to the data bank header, the index, and data series records respectively.



```
MS-DOS Prompt
Auto
C:\usmods>BCHECK KLEINBNK

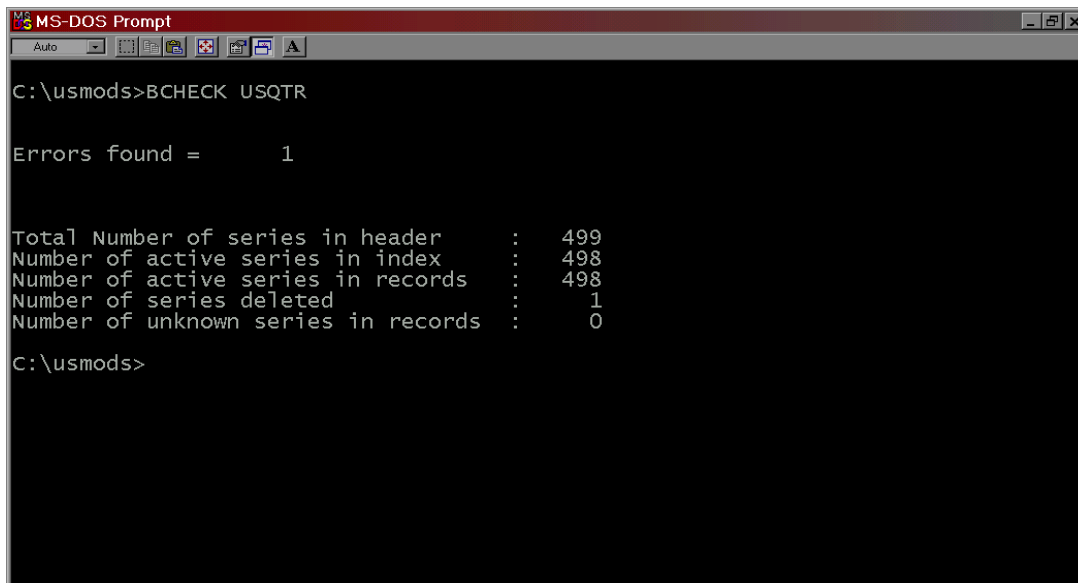
Errors found =      15

Total Number of series in header      :    15
Number of active series in index      :    15
Number of active series in records    :    15
Number of series deleted               :     0
Number of unknown series in records   :     0

C:\usmods>
```

Figure 7. BCHECK Summary Output

A contrasting characteristic of USQTR.BNK, illustrated in Figure 8, is that it contains a deleted series. If a series has been created and then deleted, it remains in a data bank. The deletion operation involves simply turning the series “off.” If you wish, it can later be restored to use. Deleted series are therefore not, per se, an indication of a corrupted data bank, but they do count in terms of the number of records used. The total number of data bank records in use consist of the sum of active series records, deleted series records, and unknown series records, the latter consisting of not yet filled records, thus not recognized as being valid.



```

MS-DOS Prompt
Auto
C:\usmods>BCHECK USQTR

Errors found =      1

Total Number of series in header      : 499
Number of active series in index      : 498
Number of active series in records    : 498
Number of series deleted               : 1
Number of unknown series in records   : 0

C:\usmods>

```

Figure 8. USQTR Summary Report

Note also that the summary display shown in Figure 8 indicates that the USQTR.BNK contains 1 error. Whenever apparent errors are found in a bank, BCHECK automatically generates, in addition to the summary report, a file called bankname.LOG, where “bankname” is the name of the bank being checked. Figure 9 displays a portion of USQTR.LOG.

The use of the word “apparent” in the last paragraph to describe the errors reflects the fact that USQTR is a data bank created for use with MODLER and DATAVIEW versions *since version 5.20*. BCHECK should be used *only* for versions before this, in part because it treats as “invalid” observations any observations containing the modern NA code value $-1E11$. As explained in the main Introduction, MODLER, MODLER BLUE, and DATAVIEW each allow for the possibility that a data series may be missing observation values for any of a variety of reasons. Such missing values are assigned a specific numeric value, which is today the value $-1E11$ (-1×10^{11}). However, originally, in 1969, the corresponding NA value assigned was -9999.99 . Versions of MODLER, MODLER BLUE and DATAVIEW prior to 5.20 used the old value for NA. All versions from 5.20 on create banks with the new NA value, which includes all Windows versions of these programs

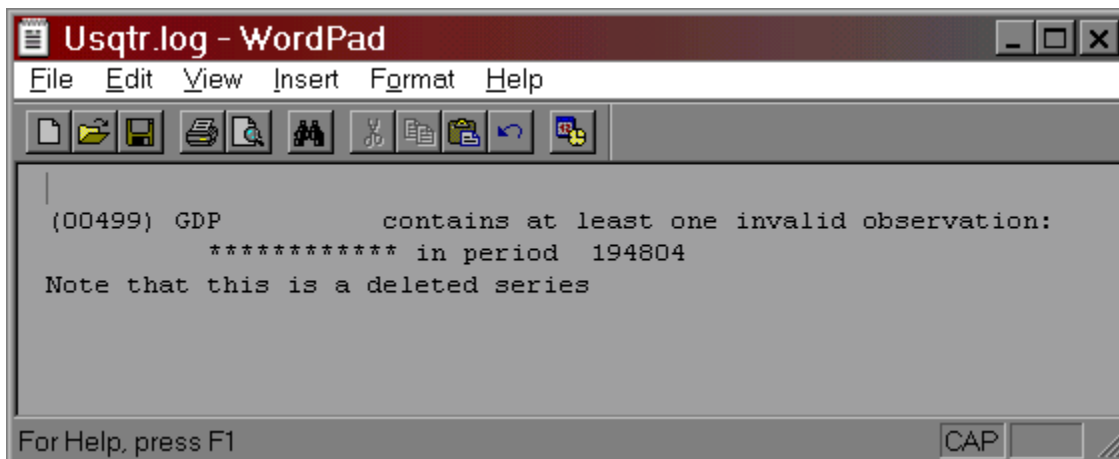


Figure 9. USQTR.LOG file

This example has been used in order to demonstrate the limitation of the BCHECK program to what are now relatively early versions of MODLER, MODLER BLUE , and DATAVIEW.

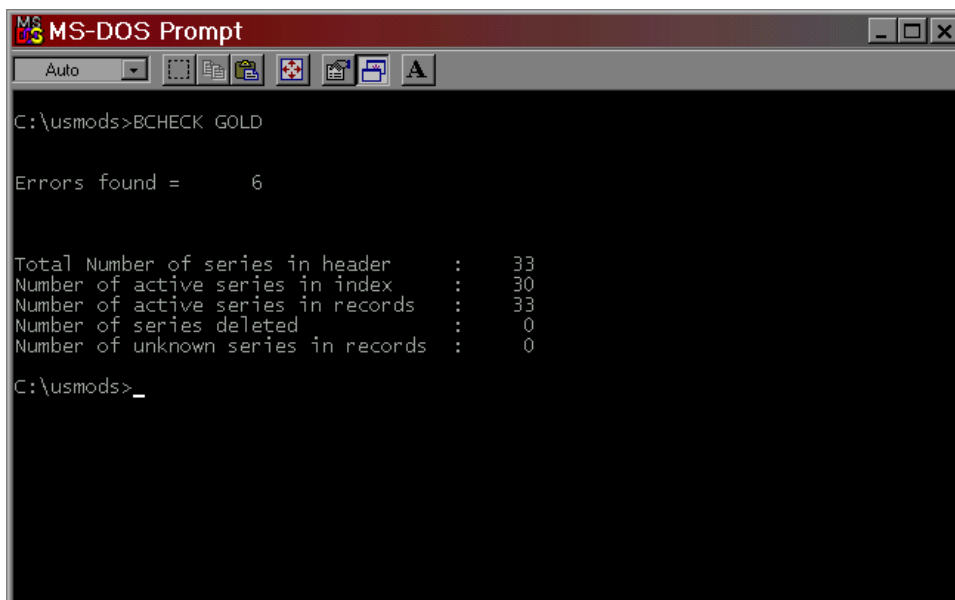
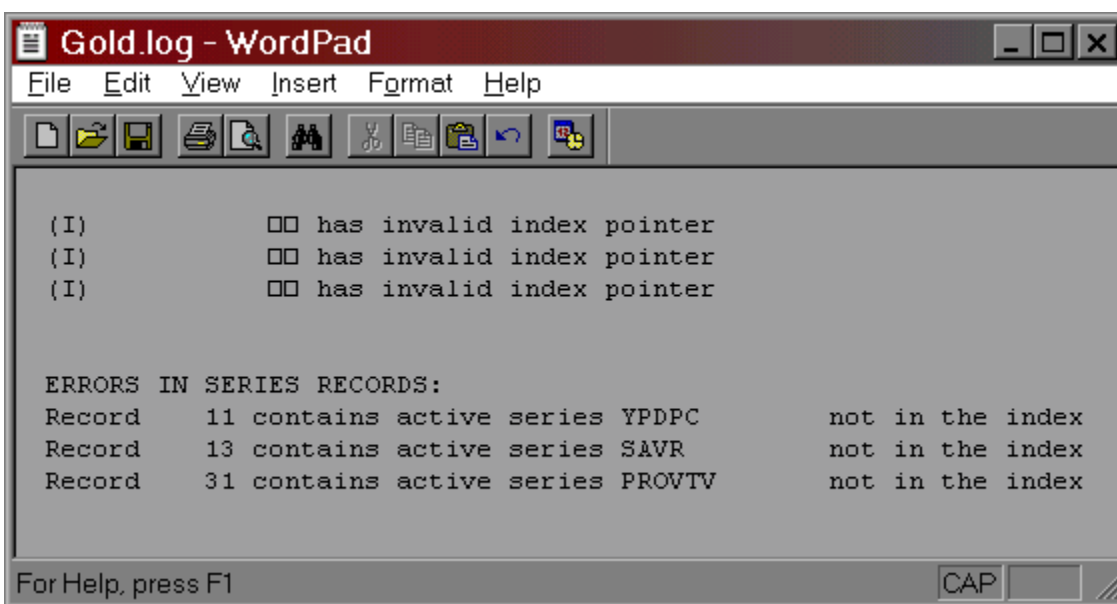


Figure 10. An Example of a Corrupted Data Bank

In contrast to Figure 8, Figure 10 just above illustrates an instance of a bank that contains actual errors. Note that the number of active series in the index is less than the number of active series records. One of the implications of this type of data bank error is that certain of the series in the data bank cannot be used. A series is retrieved for use by name and in order to do this that series name must be included in the data bank index. It is also significant that the number of unknown series is 0. An “unknown series” in this

context is a corrupted series record, so that we know that the missing series in the index are nonetheless ostensibly valid series, so far as their record representation is concerned. The question now is how to determine what the particular problem is?

The second level of error analysis is performed using the error log file, a portion of which is displayed in Figure 11. As explained earlier, whenever the BCHECK program discovers data bank errors, it automatically creates a log file, called bankname.LOG, where “bankname” is the name of the bank being analyzed, in this case GOLD.BNK. This log provides a detailed description of the errors and can be viewed using almost any text editing or word processing program. In the Windows context, Notepad and/or WordPad are immediately available to use for this purpose.



```

Gold.log - WordPad
File Edit View Insert Format Help
(I)      [ ] has invalid index pointer
(I)      [ ] has invalid index pointer
(I)      [ ] has invalid index pointer

ERRORS IN SERIES RECORDS:
Record   11 contains active series YPDPC      not in the index
Record   13 contains active series SAVR       not in the index
Record   31 contains active series PROVTV     not in the index

For Help, press F1
CAP

```

Figure 11. GOLD.LOG File

Why this type of error should occur is difficult to determine after the fact, but the nature of what has occurred is evident: the index has three elements that each have an invalid index pointer. Moreover, the fact that no series name appears in each of these cases implies that the problem is really that the index elements are effectively empty. The symbols you see instead of series names are extraneous ASCII box characters, rather than the alphabetic characters that might be expected. Therefore it is almost irrelevant that the index pointers do not point at some existing series.

Further, more pertinent information is provided by the text that follows ERRORS IN SERIES RECORDS: Ostensibly, records 11, 13, and 31 contain valid data series and these are apparently the ones that are missing from the data bank index. In fact, the inference that can be drawn from Figure 11 is that GOLD.BNK might be reconstructable for the series records all appear to be complete. The problem seems to be limited to the bank index.

The following steps should be performed: first, using MODLER, MODLER BLUE, or DATAVIEW, create a brand new data bank. Call it (for example) TEMP.BNK. Next, close all banks. Then re-open TEMP.BNK as a **STORAGE** bank and GOLD.BNK as an **ACCESS** bank. Once you have done this, issue the command:

```
COPY SERIES=1-33 FROM GOLD TO TEMP
```

The logic here is that we know, from Figure 10, that the number of active series in GOLD.BNK is 33. Furthermore, we know that the index to this bank is corrupted, so that we do not want to use that index in any way. Thus the choice of 1-33, which tells MODLER (or MODLER BLUE or DATAVIEW) to copy the first 33 series records from GOLD.BNK. During the copy process, as a matter of course, a new index will be built for the bank TEMP.BNK. The net result should be the creation of a bank, TEMP.BNK, that both contains the 33 series and has a good index. After you have saved a backup of GOLD.BNK, you can then rename TEMP.BNK to GOLD.BNK. Finally, run ANALYZE.EXE again to verify that the new GOLD bank is error-free.

As the foregoing discussion illustrates, the BCHECK program has been created in order to identify data bank errors. Sometimes, as when the individual series records are complete, these errors can be corrected. Once you have identified that a bank contains compositional errors you can contact your technical support representative, who may be able to tell you immediately how to solve the problem. In other instances, the errors may take more effort to correct.

The BCHECK program is designed to be quite sensitive to potential data bank problems and can in some cases indicate that a bank contains errors even when the bank is, to all intents and purposes, perfectly usable. There are certain data bank “errors” that are cosmetic. This is only to say that there are some rules of data bank construction that can be broken without affecting the use of a bank. However, there are no reported cases that BCHECK has found a bank to be error free that the bank then has not worked properly.

PROBE.EXE

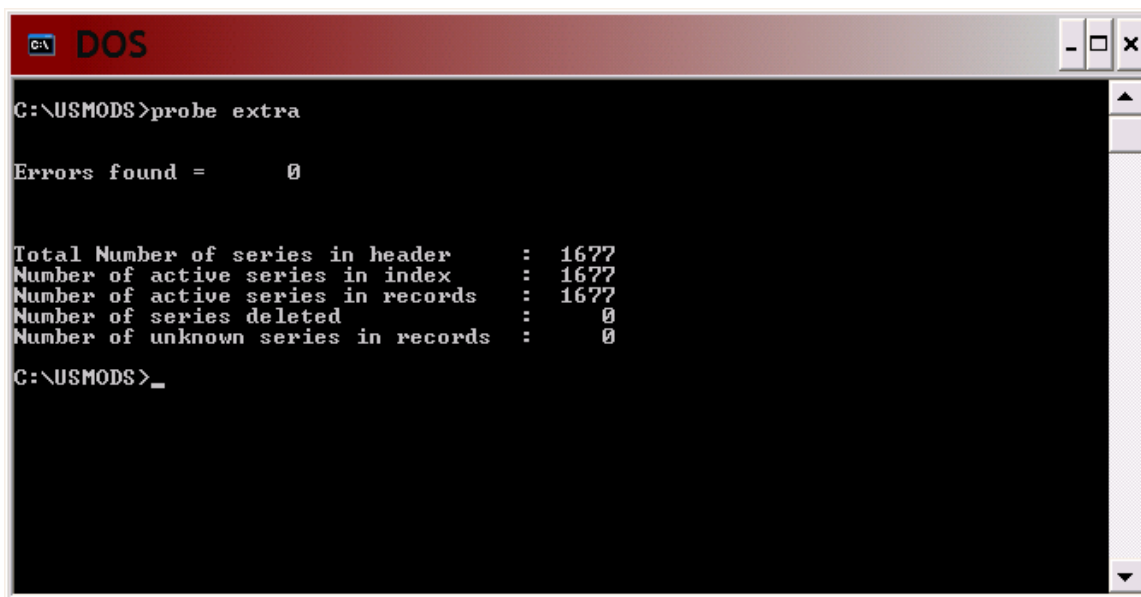
Data Bank Validation Program

Like ANALYZE, the PROBE program evaluates a data bank, checking to see if it has any compositional errors, such as data series that are not properly indexed or that otherwise exhibit some pathology. The program is run using the syntax:

PROBE bankname

where bankname is the name of a MODLER, MODLER BLUE, or DATAVIEW data bank. Provided that the bankname has the default extent .BNK, this extent can be omitted when issuing the PROBE execution command.

Figure 12 provides an example of the summary output of this program if no data bank errors are found. Note that this display states the number of series in the bank according to the data bank header, the index, and data series records respectively. One of the ways in which a data bank can become corrupted is to loose all or part of its index, or for the number of series as recorded in the header to cease for some reason to match the number of series records.



```

C:\USMODS>probe extra

Errors found =      0

Total Number of series in header      : 1677
Number of active series in index      : 1677
Number of active series in records    : 1677
Number of series deleted               : 0
Number of unknown series in records   : 0

C:\USMODS>_

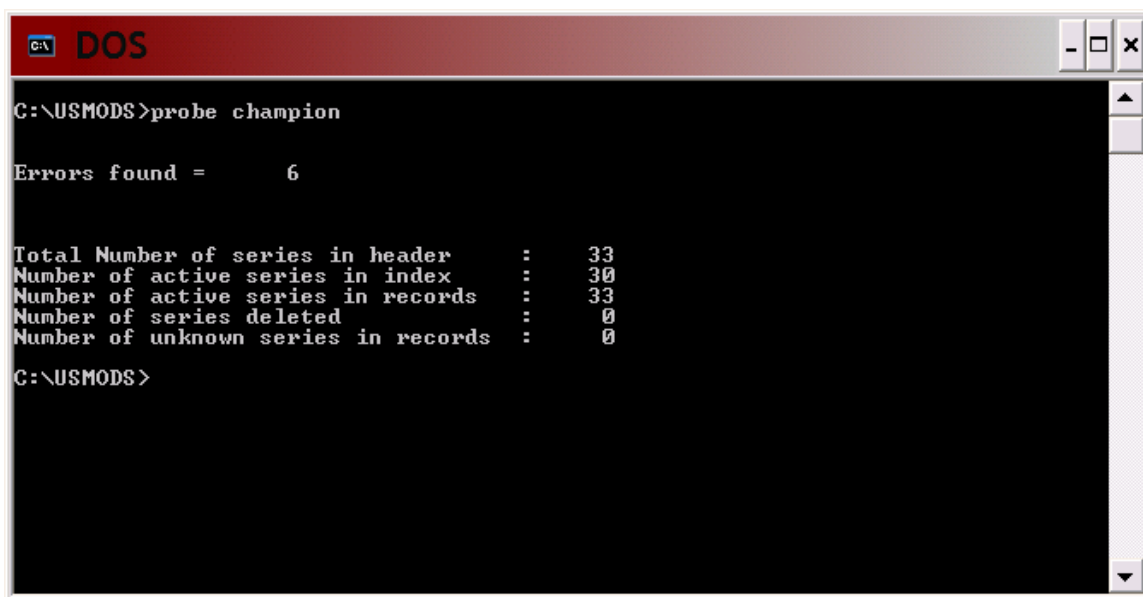
```

Figure 12. PROBE program summary output

The information given in Figure 12 is not all error-related. For instance, this bank happens not to contain any deleted series. But, by design, if a series is created and then deleted, it remains in a data bank. The deletion operation involves simply turning the series “off.” If needed, such a series can later be restored to use. The presence of

deleted series is therefore not, per se, an indication of a corrupted data bank, but such series do count in terms of the number of records used. The total number of data bank records available consist of the sum of active series records, deleted series records, and unknown series records, the latter consisting of as yet unused, but allocated records which therefore are not recognized as valid series.

In contrast to Figure 12, Figure 13 illustrates an instance of a bank that contains errors, in this case correctable. This bank was previously considered using the ANALYZE program earlier. Note that the number of active series in the index is less than the number of active series records. One of the implications of this type of data bank error is that certain of the series ostensibly in the data bank cannot be used. A series is retrieved for use by name and in order to retrieve it that series' name must be present in the data bank index. The question this circumstance raises is: how to determine what the specific problem is? It would be useful to know, for example, which series appear in the records that do not appear in the index. Ideally, it would also be nice to know why the problem has occurred, although in isolated cases this is generally very difficult to determine after the fact.



```

C:\USMODS>probe champion

Errors found =      6

Total Number of series in header      :    33
Number of active series in index      :    30
Number of active series in records    :    33
Number of series deleted               :     0
Number of unknown series in records   :     0

C:\USMODS>

```

Figure 13. Example of a Corrupted Data Bank

As in the case of the ANALYZE program, when a bank exhibits one or more errors, the second level of error analysis is performed using the error log file. Whenever the PROBE program discovers data bank errors, it automatically creates a log file, called bankname.LOG, where “bankname” is the name of the bank being probed, in this case CHAMPION.BNK. This log provides a detailed description of the errors and can be viewed using almost any text editing or word processing program. In the Windows context, Notepad and/or WordPad are immediately available to use for this purpose. Figure 14 displays a portion of the file CHAMPION.LOG in the context of Wordpad.

```

Header Record Values:
Bank Type Code:      99
Base Date and Frequency: 194701      12
Number of Active Series in Bank:      33
Number of Last Series Record:         33
Maximum Number of Series Possible:    10000
First Index Record:      37
Last Index Record:      37
Record Length:          660      174
Number of and Last Public Series Records:      0
Index Vector:          33 10000      25      0      500      20      0      660
Special Index Vector:      99 10000      660      33      33 -1907      2
Number of Index Labels:      33
Max Number of Index Labels per record:      174
Frequency Code (Alternative):      12
Number of Index-to-Index Records:      1

Specific Issues (if any):

(I)      [] has invalid index pointer
(I)      [] has invalid index pointer
(I)      [] has invalid index pointer

ERRORS IN SERIES RECORDS:
Record   11 contains active series YPDPC      not in the index
Record   13 contains active series SAVR      not in the index
Record   31 contains active series PROVTV      not in the index

For Help, press F1

```

Figure 14. CHAMPION.LOG Error File

Comparing this display with that previously shown in Figure 3 above, you will immediately see that the log file generated by the PROBE program provides a considerable amount of additional information. Some of this information is confirmatory, such as the Bank Type code, which should always be 99. However, this display also tells you the base observation frequency for the data series, here 12, and the base date of the bank, here 194701, which is the earliest date of any series that the bank could contain. The fact that the last series record number, 33, is equal to the number of active series in the bank indicates that no series have been deleted. The record length, 660, is the maximum number of observations that any single series in the data bank can contain. The record length 174 is the maximum possible number of series labels in any index record; this record length is also indicated 5 lines later. The “index vector” and “special index vector” values are there in case you wish to work with your Alphametrics support person to figure out more detailed characteristics of this bank; for the moment these numbers can be ignored. The number of index records, here shown as 1 (and

called “index to index records”) reflects that as many as 174 labels can be stored per index record, well in excess of the actual number of labels for the series in this bank, 33. Consequently, only one index record is needed for the present bank. None of these information lines indicate a problem with the bank.

In fact, the cause of the problem that Figure 14 illustrates is difficult to determine ex post, but what has occurred symptomatically is that the index has three elements that each have an invalid index pointer. Actually, the fact that no series name appears in each of these cases implies that the problem is really that the index elements are effectively empty. Therefore it is almost irrelevant that the index pointers do not point at some existing series. The symbols that you see in Figure 14 instead of series names are extraneous ASCII box characters, rather than the alphabetic characters that might be expected. Evidently, the series names were at some point somehow overwritten by “garbage.”

Further information is provided by the text that follows: ERRORS IN SERIES RECORDS. Ostensibly, records 11, 13, and 31 contain valid data series and these are apparently the ones that are missing from the data bank index. In fact, the inference that can be drawn from Figure 14 is that CHAMPION.BNK might be reconstructable: the series records all appear to be complete. The problem seems to be limited to the bank index.

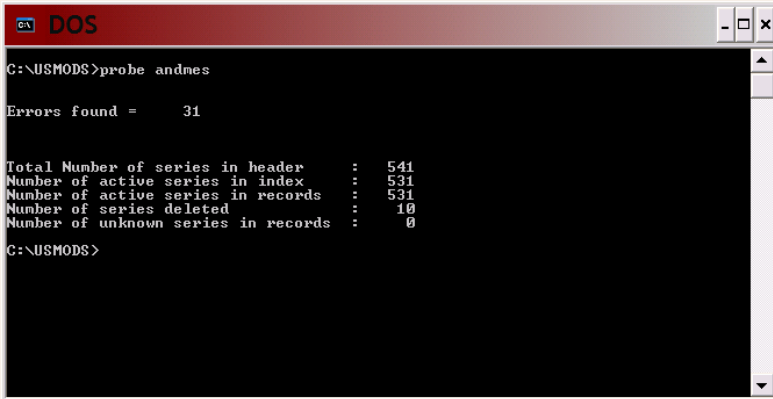
The following can be done: first, using MODLER, MODLER BLUE, or DATAVIEW, create a brand new data bank. Call it (for example) TEMP.BNK. Next, close all banks. Then re-open TEMP.BNK as a **STORAGE** bank and CHAMPION.BNK as an **ACCESS** bank. Once you have done this, issue the command:

```
COPY SERIES=1-33 FROM CHAMPION TO TEMP
```

The logic here is that we know, from Figure 13, that the number of active series in CHAMPION.BNK is 33. Furthermore, we know that the index to this bank is corrupted, so that we do not want to use that index in any way. Thus the choice of 1-33, which tells MODLER (or MODLER BLUE or DATAVIEW) to copy the first 33 series records from CHAMPION.BNK. During the copy process, as a matter of course, a new index will be independently built for the bank TEMP.BNK. The net result should be the creation of a bank, TEMP.BNK, that both contains the 33 series and has a good index. After you have saved a backup of CHAMPION.BNK, you can then rename TEMP.BNK to CHAMPION.BNK. Finally, run PROBE.EXE again to verify that the new CHAMPION bank is error-free.

Now consider the result shown in Figure 15. The fact that the total number of series in the header, 541 is greater than the number of active series in the index and in records, 531, is not per se a problem, for note that 10 series have been deleted, thus explaining the difference. However, there are nonetheless 31 errors found in this bank.

To explain the nature of these errors, we again need to look at the log file, here called ANDMES.LOG. It can be displayed using the WordPad program found on any Windows machine, as illustrated in Figure 16.



```

C:\USMODS>probe andmes

Errors found =      31

Total Number of series in header      : 541
Number of active series in index      : 531
Number of active series in records    : 531
Number of series deleted               : 10
Number of unknown series in records   : 0

C:\USMODS>
```

Figure 15. Bank With Deleted Series

```

ANDMES - WordPad
File Edit View Insert Format Help
Header Record Values:
Bank Type Code: 99
Base Date and Frequency: 195001 12
Number of Active Series in Bank: 541
Number of Last Series Record: 531
Maximum Number of Series Possible: 10000
First Index Record: 546
Last Index Record: 595
Record Length: 732 192
Number of and Last Public Series Records: 0
Index Vector: 50 10000 25 0 500 20 0 732
Special Index Vector: 99 10000 732 541 531 -1607 2
Number of Index Labels: 531
Max Number of Index Labels per record: 192
Frequency Code (Alternative): 12
Number of Index-to-Index Records: 50

Specific Issues (if any):

(00540) CCEADM contains no observations
(00538) IPH units contains control characters: 0000
(00505) IVUM units contains control characters: 0000
(00506) IVUMCA units contains control characters: 0000
(00507) IVUMCO units contains control characters: 0000
(00508) IVUMIN units contains control characters: 0000
(00509) IVUMR units contains control characters: 0000
(00510) IVUMUE units contains control characters: 0000
(00511) IVUX units contains control characters: 0000
(00512) IVUXCA units contains control characters: 0000
(00513) IVUXCO units contains control characters: 0000
(00514) IVUXIN units contains control characters: 0000
(00515) IVUXR units contains control characters: 0000
(00516) IVUXUE units contains control characters: 0000
(00327) NIPC.ALISELA units contains control characters: 0000
(00327) NIPC.ALISELA source contains control characters: 00000000
(00537) NUMHIPO units contains control characters: 0000
(00524) TCERUE units contains control characters: 0000
(00287) IPIANV units contains control characters: 0000
(00287) IPIANV source contains control characters: 00000000
Note that this is a deleted series
(00288) IPIANV21 units contains control characters: 0000
(00288) IPIANV21 source contains control characters: 00000000
Note that this is a deleted series
(00289) IPIANV22 units contains control characters: 0000
(00289) IPIANV22 source contains control characters: 00000000
Note that this is a deleted series
(00290) IPIANV23 units contains control characters: 0000
(00290) IPIANV23 source contains control characters: 00000000
Note that this is a deleted series
(00346) NIPC.NPROENE units contains control characters: 0000
(00346) NIPC.NPROENE source contains control characters: 00000000
Note that this is a deleted series
(00439) SONETMA units contains control characters: 0000
(00439) SONETMA source contains control characters: 00000000
For Help, press F1

```

Figure 16. ANDMES.LOG file displayed

Certain of the problems with this bank are essentially cosmetic. For example, the fact that the Units and Source codes contain control characters. “Control” characters are ASCII characters that may be unprintable, create sounds, or display in the form of funny faces. In general, text fields like the Units and Source code fields should contain only text characters.

Another example of a bank that contains errors is illustrated by Figure 4. These errors are not, however, evident in the number of series in the header, index, or records, but instead are more specific to the individual series records. We infer this from the fact that the index, the number of active series, and the number of records all match. At the same time, there are still apparently 17005 errors in the bank.

```

MS-DOS Prompt
Auto
C:\usmods>ANALYZE ANNUAL

Errors found = 17005

Total Number of series in header      : 6977
Number of active series in index      : 6977
Number of active series in records    : 6977
Number of series deleted               : 0
Number of unknown series in records   : 0

C:\usmods>_

```

Figure 17. Analysis of ANNUAL.BNK

Figure 5 displays only a small portion of ANNUAL.LOG. However, this is enough to tell us the general nature of the problem. From the top working down, note first at the left the bracketed number (00001), which refers to the series record number 1. Each of the lines that begin (00001) refer to this series. Seemingly, it is named AA and has something to do with “Auto Output” which is stated in Billions of \$, as implied by the bracketed (Bil.\$). The units code is US\$, obviously for US dollars. The source code is BEA, for the Bureau of Economic Analysis. However, there are a lot of extraneous boxlike characters, but the real give away is the notation in each case the element “contains control characters.” What has occurred is that whoever originally created the bank, which was probably done using a huge macro file, created the macro file with a text or word processing package that did not produce simple ASCII text.

```

(H) Bank has invalid type:      0
(00001) AA      description contains control characters:
      Auto Output (Bil.$)
(00001) AA      units contains control characters: US$
(00001) AA      source contains control characters: BEA
(00002) AANUA   description contains control characters:
      Ward's Auto Assemblies (NSA, Thous.Units)
(00002) AANUA   units contains control characters: US$
(00002) AANUA   source contains control characters: BEA
(00003) AB      description contains control characters:
      Total Bankruptcy Filings, U.S. (Units)
(00003) AB      source contains control characters: US$
(00004) ABCSUA  description contains control characters:
      Auto Imports from Canada, BEA (SA, Thous.Units)
(00004) ABCSUA  units contains control characters: US$
(00004) ABCSUA  source contains control characters: BEA
(00005) ABMSUA  description contains control characters:
      Auto Imports from Mexico, BEA (SA, Thous.Units)
(00005) ABMSUA  units contains control characters: US$
(00005) ABMSUA  source contains control characters: BEA
(00006) ABPRSUA description contains control characters:
      Auto Production, BEA (SA, Thous.Units)
(00006) ABPRSUA units contains control characters: US$
(00006) ABPRSUA source contains control characters: BEA
(00007) ABPSUA  description contains control characters:
      BEA Auto Production Estimate (SA, Mil.Units)
(00007) ABPSUA  units contains control characters: US$
(00007) ABPSUA  source contains control characters: BEA
(00008) ABXSUA  description contains control characters:
      Exports of Autos, BEA (SA, Thous.Units)
(00008) ABXSUA  units contains control characters: US$
(00008) ABXSUA  source contains control characters: BEA
(00009) ACA     description contains control characters:
      Auto Output (Bil.Fxd.96$)
(00009) ACA     units contains control characters: US$
(00009) ACA     source contains control characters: BEA
(00010) ACNA    description contains control characters:
      PCE: New Autos (Bil.$)
(00010) ACNA    units contains control characters: US$
(00010) ACNA    source contains control characters: BEA
(00011) ACNCA   description contains control characters:
      Autos, Real New: Personal Consumption Expenditures (SA, Bil.Fxd.1992$)
(00011) ACNCA   units contains control characters: US$
(00011) ACNCA   source contains control characters: BEA
(00012) ACNDHUA description contains control characters:
      PCE: New Domestic Autos (SA, Mil.Chn.1996$)
(00012) ACNDHUA units contains control characters: US$
  
```

Figure 18. ANNUAL.LOG

Control characters, as a phenomenon, consist of visually unrepresentable ASCII character codes. codes for tab spaces, line feeds, carriage returns, and other such things that instruct printers, CRTs and other devices, telling them how to display text. However, these codes, if they get copied into a data bank, generally gum up the works. At minimum, the problem is that what may appear to be a single space (the equivalent of a single blank or other character on the screen) might be a tab character, which can cause printed output to lurch crazily across a page. Or the character can be a “bell” character, which when encountered displaying or reading the file, causes the computer’s bell to sound. Alternatively, the appearance of such characters can cause series names to

become unrecognizable: the computer does not necessarily interpret AA followed by a blank as being the same thing as AA followed by a bell character or a tab character.

ANNUAL.BNK is actually a marginally usable bank. For example, if you look at Figure 6, you will see that it displays the series AA. Note that the description, source, and units of this series are properly displayed, and that generally nothing seems to be amiss. The problem occurs in other contexts. For instance, attempts to display portions of the data bank index result in error messages to the effect that there are “Too many series names found in the search interval.” Particularly when banks, such as this one, grow to 6000 or more series in size it is important to be able to deal with series in bulk.

Series: AA Source: BEA Units: US\$ Available: 1947-1999

Description: Auto Output (Bil.\$)

Last Extended: 7/28/ 0 Last Revised: 10/27/00

Annual Data

	I	II	III	IV	V	VI
1947-1952	7,2000	8,8000	11,900	15,400	13,300	12,000
1953-1958	16,100	14,700	21,200	16,900	19,400	14,400
1959-1964	19,400	21,400	17,700	22,400	25,100	25,900
1965-1970	31,300	30,300	27,700	35,200	34,900	28,800
1971-1976	39,200	41,200	46,300	39,800	40,700	55,600
1977-1982	65,200	69,700	68,300	60,500	70,600	65,700
1983-1988	88,300	103,800	114,200	118,800	115,200	121,600
1989-1994	121,800	113,100	102,300	111,700	121,900	133,300
1995-2000	130,500	126,100	126,700	127,300	126,100	

Print OK

Figure 19. Print Out of Series AA from ANNUAL.BNK

As the foregoing discussion illustrates, the PROBE program has been created in order to identify data bank errors. Sometimes, as when the individual series records are complete, these errors can be corrected. Once you have identified that a bank contains compositional errors you can contact your technical support representative, who may be able to tell you immediately how to solve the problem. In other instances, the errors may take more effort to correct.

The PROBE program is designed to be quite sensitive to potential data bank problems and can in some cases indicate that a bank contains errors even when the bank is, to all intents and purposes, perfectly usable. There are certain data bank “errors” that are cosmetic. This is only to say that there are some rules of data bank construction that can

be broken without affecting the use of a bank. However, there are no reported cases that PROBE has found a bank to be error free that the bank then has not worked properly.

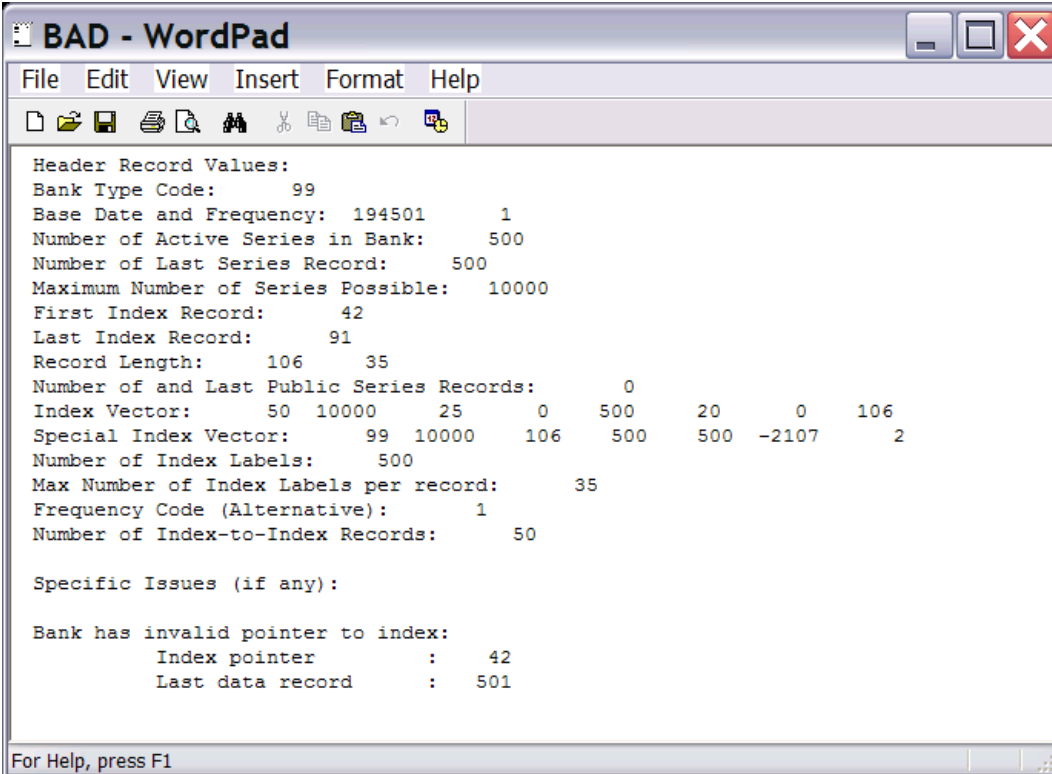
A further type of problem is shown in Figure xx.



```
C:\USMODS>probe bad

FATAL ERROR(S) IN BANK HEADER

C:\USMODS>_
```



```
BAD - WordPad
File Edit View Insert Format Help
Header Record Values:
Bank Type Code:      99
Base Date and Frequency: 194501      1
Number of Active Series in Bank:      500
Number of Last Series Record:      500
Maximum Number of Series Possible: 10000
First Index Record:      42
Last Index Record:      91
Record Length:      106      35
Number of and Last Public Series Records:      0
Index Vector:      50 10000      25      0      500      20      0      106
Special Index Vector:      99 10000      106      500      500 -2107      2
Number of Index Labels:      500
Max Number of Index Labels per record:      35
Frequency Code (Alternative):      1
Number of Index-to-Index Records:      50

Specific Issues (if any):

Bank has invalid pointer to index:
Index pointer      :      42
Last data record   :      501

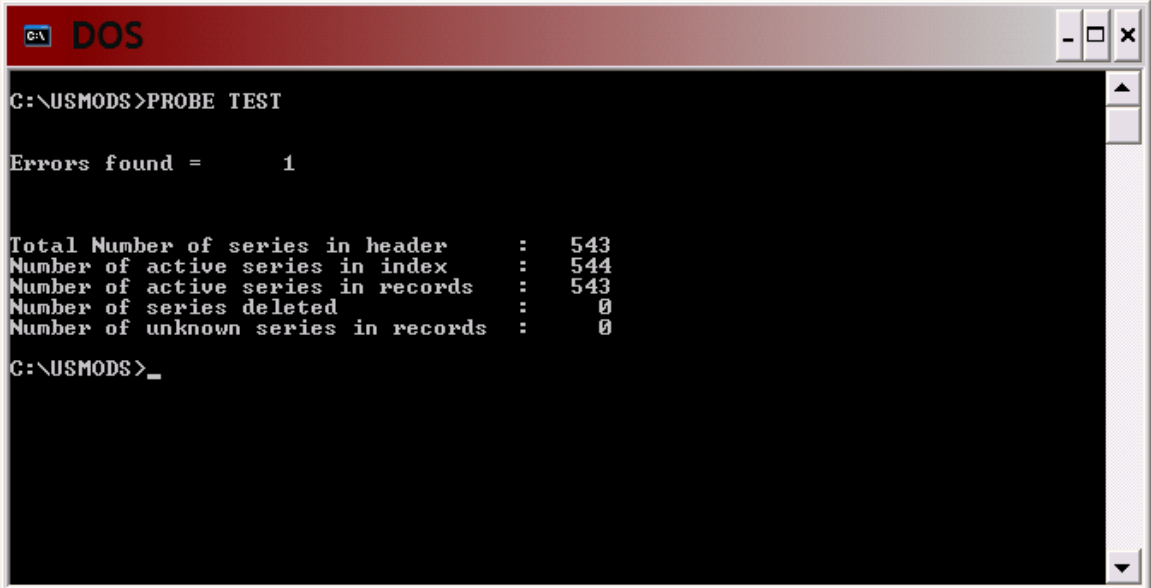
For Help, press F1
```

PROBE.EXE

Data Bank Validation Program

PROBE.EXE is an alternative to ANALYZE.EXE, described above. The fundamental difference between ANALYZE and PROBE is that PROBE provides additional, somewhat more esoteric, but possibly quite revealing information about a data bank. In particular, if at least one error is discovered in the bank, as indicated for example in Figure XX below, and if you then display the PROBE Log File, this log file will specify values that are found in the data bank header and other records. Generally, you are likely to need then to talk to your support person at Alphametrics to determine which of these values sheds light on the specific problem encountered.

For example, you will see in Figure XX below that the PROBE program is executed in DOS mode using the word PROBE followed by the name of a bank. Here, the bank in question is called TEST.BNK, although as illustrated the BNK extent does not need to be specified.



```
C:\ DOS
C:\USMODS>PROBE TEST

Errors found =      1

Total Number of series in header      : 543
Number of active series in index      : 544
Number of active series in records    : 543
Number of series deleted               : 0
Number of unknown series in records   : 0

C:\USMODS>_
```

The associated log file is shown in Figure XY.

TEST - WordPad

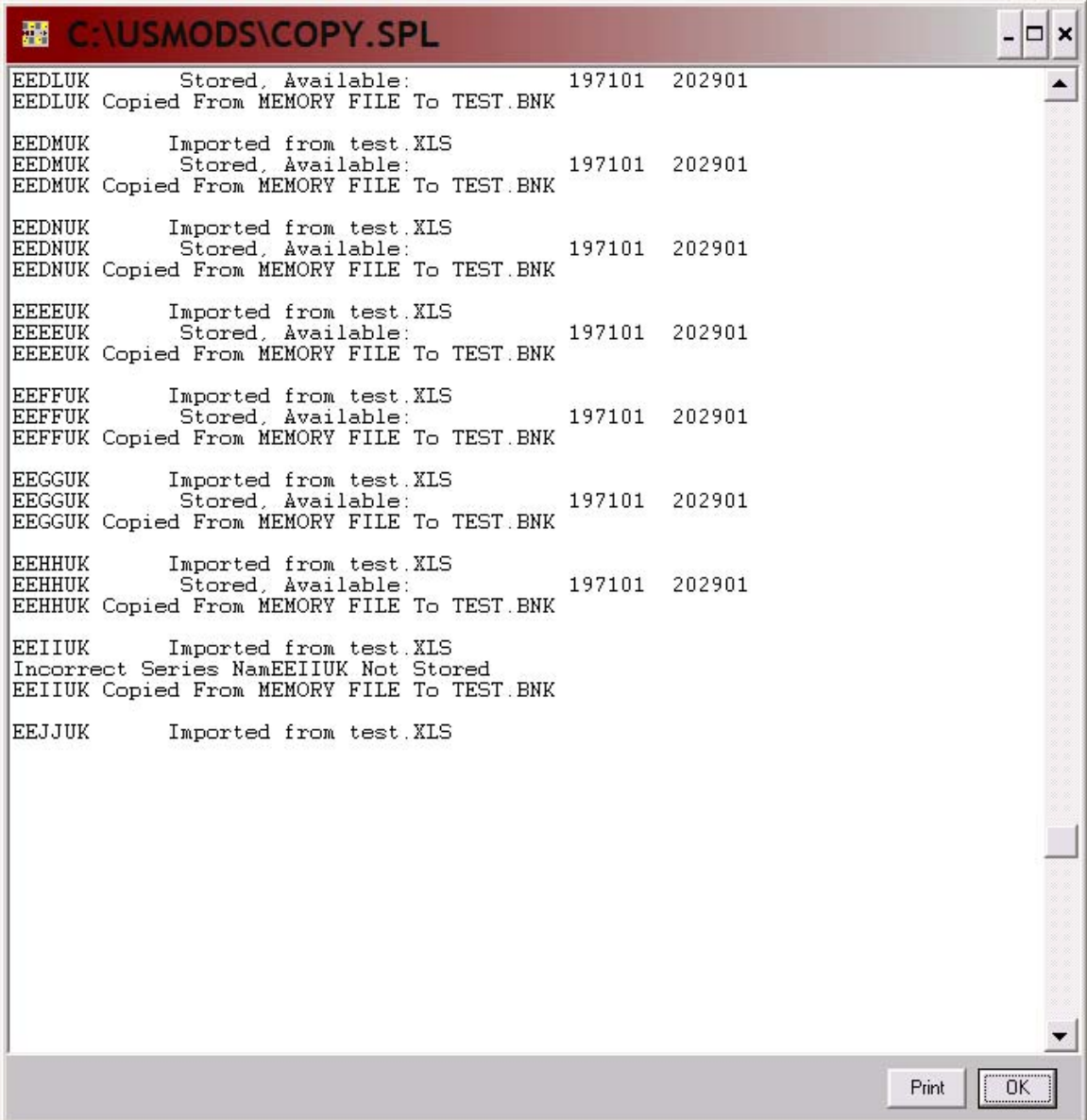
File Edit View Insert Format Help

Header Record Values:
 Bank Type Code: 99
 Base Date and Frequency: 194501 1
 Number of Active Series in Bank: 543
 Number of Last Series Record: 543
 Maximum Number of Series Possible: 10000
 First Index Record: 546
 Last Index Record: 595
 Record Length: 106 35
 Number of and Last Public Series Records: 0
 Index Vector: 50 10000 25 0 500 20 0 106
 Special Index Vector: 99 10000 106 543 543 -2107 2
 Number of Index Labels: 543
 Max Number of Index Labels per record: 35
 Frequency Code (Alternative): 1
 Number of Index-to-Index Records: 50

Specific Issues (if any):

(I) EEIIUK has index pointer past end of data records

For Help, press F1



```
C:\USMODS\COPY.SPL
EEDLUK      Stored, Available:      197101  202901
EEDLUK Copied From MEMORY FILE To TEST.BNK

EEDMUK      Imported from test.XLS
EEDMUK      Stored, Available:      197101  202901
EEDMUK Copied From MEMORY FILE To TEST.BNK

EEDNUK      Imported from test.XLS
EEDNUK      Stored, Available:      197101  202901
EEDNUK Copied From MEMORY FILE To TEST.BNK

EEEEUK      Imported from test.XLS
EEEEUK      Stored, Available:      197101  202901
EEEEUK Copied From MEMORY FILE To TEST.BNK

EEFFUK      Imported from test.XLS
EEFFUK      Stored, Available:      197101  202901
EEFFUK Copied From MEMORY FILE To TEST.BNK

EEGGUK      Imported from test.XLS
EEGGUK      Stored, Available:      197101  202901
EEGGUK Copied From MEMORY FILE To TEST.BNK

EEHHUK      Imported from test.XLS
EEHHUK      Stored, Available:      197101  202901
EEHHUK Copied From MEMORY FILE To TEST.BNK

EEIIUK      Imported from test.XLS
Incorrect Series NamEEIIUK Not Stored
EEIIUK Copied From MEMORY FILE To TEST.BNK

EEJJUK      Imported from test.XLS
```


CDOC.EXE

Global Editing of Data Bank Documentation

CDOC.EXE is a utility program that can be used to globally edit the documentation of MODLER data bank series. It can be used to edit documentation within a single bank, or to selectively copy series documentation from one data bank to another. One of the uses for CDOC is to assist you in creating a bank corresponding to a particular econometric model, particularly when the model has been created using data from a variety of banks.

The general command syntax for CDOC is

```
CDOC bankname1 [TO bankname2] [WITH edits|*] [/V][F]
```

where "bankname1" and "bankname2" each refer to the name of a data bank. This syntax might best be illustrated by example, as will be done. However, you can use CDOC either as a standalone program or execute it as a subtask from MODLER, MODLER BLUE, or DATAVIEW. Furthermore, for certain variations in the command, you can use DOS redirection from the screen to a file.

To Scan the Documentation in a Single Bank:

In order to scan the documentation in a single bank and create a listing in the context of a file called "list", use the command syntax:

```
CDOC bankname /V > list
```

The documentation for all series in the bank will be listed in this file, as illustrated in Figure 12, using the demonstration bank KLEINBNK.BNK distributed with MODLER, MODLER BLUE and DATAVIEW. Note that, for each series, the first line displays the series name, followed by the units code, source code and date(s) of series revision and/or extension. In this example, none of the series have been revised since the bank was created, so that revision dates are not shown. The second line provides the series description. As you can verify, if you omit the "> list", CDOC will instead display the documentation on your screen. If you omit the /V from the command, CDOC will state:

```
Bank contains 15 series
```

and do nothing else.

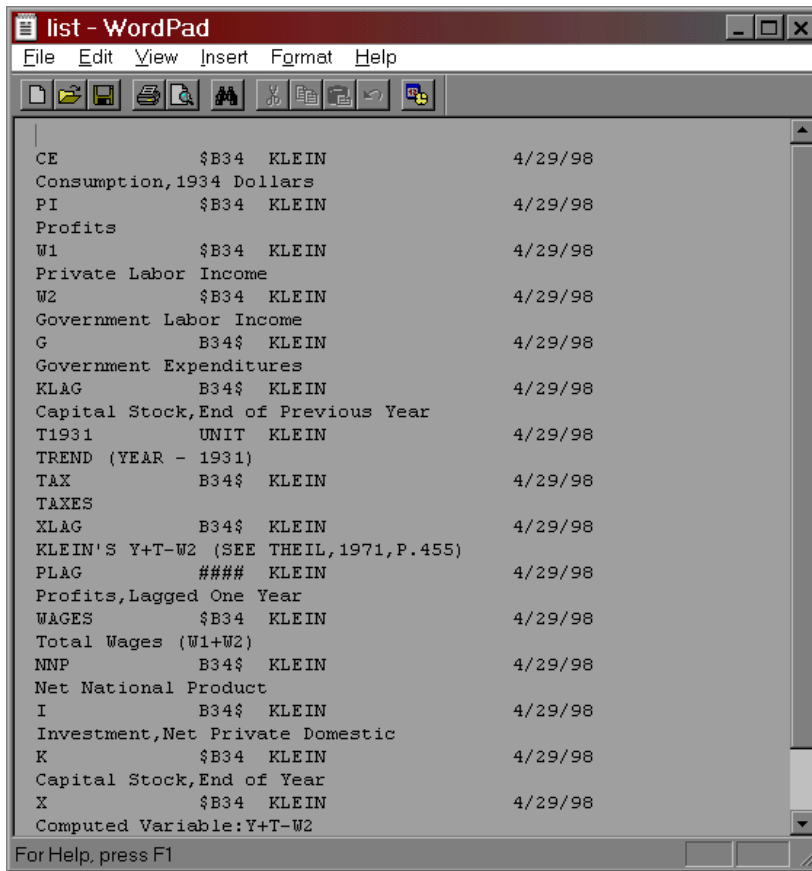


Figure 20. Onscreen Display of KLEINBNK Series Documentation

To Edit the Series Documentation for a Single Data Bank

In order to edit the documentation for a single data bank, use the general syntax:

CDOC bankname WITH editfilename /V /F

Where:

WITH is a keyword and:

bankname is the name of the bank

editfile is the name of a file containing field edit instructions

/V for verify, specifies that results are to be displayed on the screen

/F for file, specifies that results are to be written to the data bank index

Specific Edit Instructions

The specific editing instructions are written in the general form:

Opcode field text

Where both “Opcode” and fieldname keywords can be abbreviated to a single letter and “text” refers to text that you supply.

The available operations keywords are:

BLANK field
 PRESERVE field (default)
 COPY field text
 MERGE field text
 END

And the fields that may be edited are:

DESCRIPTION	(72 chars)
SOURCE	(8 chars)
UNITS	(4 chars)
REVISED (last revision date)	(8 chars)
EXTENDED (last extended date)	(8 chars)

The COPY option causes the new data to overwrite any existing data. The MERGE option writes new data only in records where the field was previously blank or N.A. END is used only when the input is from the keyboard, as will be discussed later. The default for fields not specified is PRESERVE.

The global characteristic of the editing needs to be recognized from the beginning. You can, for example, blank the SOURCE field for all series in the blank using the editing command:

BLANK SOURCE

located within the editfile. But you cannot simply replace the source field for one series, although you can perform exception processing, such as replacing ##### in series PLAG units code, as shown in Figure 12, with B34\$ using the command:

COPY UNITS B34\$

which will work because it will replace the units code for all series, only giving the *impression* that just the units code for PLAG is being replaced.

Alternatively, the syntax:

```
CDOC bankname WITH *
```

where the editfilename is replaced by the symbol *, specifies that CDOC should read edit instructions from the keyboard. For instance, if you issue the command:

```
CDOC KLEINBNK *
```

CDOC will respond with:

```
EDIT>
```

expecting you to place the editing command just to the right of the >. When you finish editing from the keyboard, issue the command END, and CDOC will immediately respond by telling you the number of series in the bank. If you instead issue the CDOC command with a /V at the end, upon your issuing the END command, CDOC will display the documentation of all series on the screen.

To Copy Documentation from One Bank to Another

In order to copy documentation between banks, use the syntax:

```
CDOC bankname1 TO bankname2 WITH editfilename /V /F
```

In this case the edit instructions determine whether and how documentation fields are copied into the destination bank. But note that editing and displays are confined to series whose names are found in *both* banks.

To Scan Documentation for Matching Series Without Copying Between Banks

To scan documentation in order to discover series present in both banks, without copying between banks, use the syntax:

```
CDOC bank TO bank /V
```

If the /V switch is omitted, CDOC will print a message stating the number of index entries found in each bank and the number of series names that match.

CHGNA.EXE and UNCHGNA.EXE

Data Bank NA Value Updating Program

As explained in the Introduction, MODLER, MODLER BLUE, and DATAVIEW each allow for the possibility that a data series may be missing observation values. Such missing values can occur for any of a variety of reasons. A reason may be that a particular observation has not been reported by the original data source because of confidentiality requirements in the collection of the underlying data. Another may be that certain observations are delayed in their release date as compared to others. But for whatever reason, it is common for economic data to exhibit the property of missing values.

When such values exist they need to be taken into account in a way that supports data integrity. Aggregating across series, it is generally not desirable that missing observations be given the value zero, for if this is done, aggregate values may be unknowingly computed that in one period are true cumulative values, representing the aggregate of all subcomponent elements, whereas in other periods they may represent a partial cumulant. Alternatively, when producing time plots, tables, or other displays it is usually desirable for the display to convey the overall message of the data. For instance, when a plot of a series is produced that contains one or more missing values it is often desirable for the plotted line or other element to display a gap for each missing value, rather than to distort the plot by explicitly plotting a zero or some other arbitrary value.

The use of a specific numeric value that is given exceptional treatment allows this type of adjustment to be made automatically by a computer program, while at the same time not prohibiting a user of the program from generating whatever alternative values he or she sees fit. For example, it is always possible for an individual user to generate a value that interpolates missing values, or in some other way compensates for them. Similarly, the fact that a value is recorded as NA when no value is entered for that observation in no way prevents the user from explicitly entering a value. The issue here is not what to do about missing values. It is rather how to account for missing values when they exist.

MODLER, MODLER BLUE and DATAVIEW today each assign missing values the specific numeric value $-1E11$ (-1×10^{11}), such number being arbitrary but chosen for the reason that it would never naturally be encountered as an actual observation value for a series. However, originally, in 1969, the corresponding NA value assigned was -9999.99 , a value that was seldom if ever found as an actual data value in those days, but has since become relatively common. This change in assigned NA value now more than 10 years ago does however mean that data banks created before that time are not in this respect compatible with those created subsequently.

The CHGNA.EXE program was created in order to deal with this assignment change. It works in a specific way. It is invoked using the syntax:

CHGNA bankname

where “bankname” is the name of a data bank. It reads through this bank series by series and whenever it encounters an observation value of -9999.99 it converts it to the value $-1E11$ (-1×10^{11}). This operation is the only one that CHGNA performs.

In one sense, the CHGNA program is value neutral: if for a given bank no series exhibits -9999.99 as an observation value, CHGNA will leave that bank unchanged. However, it is possible that the value -9999.99 could occur as an actual observation. For this reason, you should *not* use CHGNA.EXE on any bank that you know was created by MODLER, MODLER BLUE, or DATAVIEW subsequent to version 5.10. Inasmuch as version 5.10 of each of these programs long pre-dates any Windows version, unless you are a long time user of MODLER you are unlikely to ever need to use CHGNA. Even if you are a long time user, you are now unlikely to come across an old style bank without trying.

The program UNCHGNA is also run by stating the program name followed by the name of a bank:

UNCHGNA bankname

and is best understood as is the opposite of CHGNA. UNCHGNA converts a new type bank into an old type bank. It is included simply to allow you to reverse the CHGNA conversion process should you need to for some reason. However, quite obviously, care should be taken to insure that any old style banks you create not get into general circulation.

SLOAD.EXE

MODLER Solution File Data Extractor

The SLOAD program described here exists in order to extract data from a MODLER solution file and to store that data in an existing, given MODLER data bank that contains one or more data series having the same name as the model variables in that solution file, the same observational frequency, and a common defined date range. The bank and the solution file do not otherwise need to exhibit common characteristics. This version of the program handles solution files containing a maximum of 500 series. The data bank, model, and solution file must all be resident in the current directory.

One example of a way in which to use SLOAD is as a means of updating a model data bank using a particular model solution. However, the series matching requirements are sufficiently unrestrictive as to provide considerable latitude of use. For each download the number of series matched (which may be zero) is reported at the end. Index information for updated series is written to the bank

The command syntax to run SLOAD as a DOS command is:

```
SLOAD bankname FROM modelname:solvfilename
```

which of course implies the use of the syntax:

```
DOS SLOAD bankname FROM modelname:solvfilename
```

to run the program from within MODLER.