Banker

The *G* Databank Maker

I. What is BANKER?

Banker is a *G* data bank maker. It creates *G* data banks without going through G. It is particularly suited for making large *G* data banks, known as hashed banks. Since a hashed bank is *not* bounded by the 2500 maximum number of series associated with a normal workspace type *G*-bank. In other words,with *Banker*, a hashed-bank can literally include millions of series, and the size of your hard disk becomes the effective constraint. *Banker* makes both hashed (.hbk, .hin) and compressed (.cbk, .cin) G banks, however, users are strongly encouraged to produce the hashed bank, as it is now the standard data bank Inforum regularly creates and maintains.

II. Usage:

Syntax: BANKER [option] <data> [bank]

Options are one or more of the following option characters preceded by "-" or "/". If no option specified, hashed bank will be produced.

```
  BANK TYPE OPTIONS           OTHER OPTIONS
  -h hashed bank (default)    -b<###> issue number of BINs for hashed bank
  -c compressed bank          -v      view series names while processing
                              -d      debugging
                              -l      look at the finished bank
                              -t[XXX] take bank title
Examples:
BANKER lift.dat              :produces default bank (LIFT.HBK, LIFT.HIN)
BANKER -h lift.dat           :produces LIFT.HBK, LIFT.HIN
BANKER -b101 lift.dat        :makes the hashed bank with # of BINs = 101
BANKER -d lift.dat           :make hashed bank, debugging only
BANKER -c lift.dat           :produces LIFT.CBK, LIFT.CIN
BANKER -cv lift.dat          :and displays series names while processing
BANKER -cvtinforum lift.dat  :and takes the bank_title as 'inforum'
BANKER -ctinforum-d lift.dat :debugging, and bank_title = 'inforum'
BANKER -cdtinforum lift.dat  :debugging, and bank_title = 'inforum'
BANKER -cd lift.dat          :make compressed bank, debugging only
```

Note that normally only the very first option switch is required to be preceded with '-' or '/'. But there is a caveat with the the -t switch. If it is chosen and a bank title is attached next to it, then the next chosen option must be preceded with '-' or '/', which would otherwise be optional. Beside, if your bank title consists of more than one word, connect these words with '+' so that there will not be any space in between the words.

In light of this, you may decide to ignore -t switch altogether, in which case you will be prompted to provide a bank title once *Banker* starts running. Yet another way to input the bank

title is to bury it in brackets'{ }' and put it at the very beginning of your formatted data file. In away, brackets '{ }' indicate comments to *Banker*.

A file named BANKER.MSG will be automatically produced for each run of BANKER. It contains information on data compression status on each series as well as some other related messages (if any). The most commom message on data compression includes:

1. "No compression xxx: FIRST DIFFERENCE TOO LARGE or all-zero series.";
2. "No compression on yyy: HUGE NUMBER ENCOUNTERED."

The first simply means series XXX is an all-zero series or its largest first difference is greater than 32767. Note that for G-style compression purposes, the first difference is taken AFTER all the decimal places have been slided to the right until the series is all integers. This point may also help you understand the second message, in which case the largest first difference so calculated exceeds LONG_MAX (or 2,147,483,647), and we call "HUGE NUMBER COUNTERED", even though without sliding the decimal to the right until the series is all integers the largest first difference may not exceed LONG_MAX (or 2,147,483,647).

III. How to format input data file for BANKER

Banker handles all conventional G “data” and “matdat” input formats, which are assumed to be known to the G user. There are detailed discussion in G "help" files and in the Appendix of Clopper Almon's *The Craft of Economic Modeling.*

Banker also accepts the Inforum one-series-per-line format, which was initially developed to process very large data banks more efficiently, such as the World Tables of Social and Economic Indicators. The general idea of this format is to stock all the information about a series on one line (record), including, of course, all its observations.

Here is an example of the one-series-per-line format:

```
      gdp$  1985  Q  1  0  5025  5130.5  5.255e3

where,
     gdp$ : series_name
     1985 : baseyear (can also be written as 85)
        Q : frequency (M = Monthly, A = Annual)
        1 : starting_period
        0 : decimal point left-shift factor
     5025 : observation #1
   5130.5 : observation #2
  5.255e3 : observation #3
```

All seems clear except, perhaps, the fifth item under this format -decimal point left-shift factor. Thus some explanations (orjustifications for it) are in order. Simply put, this left-shift factor enables you to manipulate (add) the decimal places in your data. For example, some BLS

data often requires some additional decimal places, and this left-shift factor can help you do just that. If, on the other hand, there is no need to modify decimal places in your data, use zero as the left-shift factor.

Please take note that each item must be separated by at least one space (as shown in the example above). It should be mentioned in passing that *Banker* will keep intact the series names exactly as in the formatted input file. That is, if a series name (or part of it) is in upper (lower) case, it will remain that way in the bank created by *Banker*.

Finally, *Banker* allows *G*-style commenting.

Examples:

```
# This is a one line comment.
: This comment, however, is longer.
Clearly, it is a much more important comment,
For it takes three lines.
: Or four.
```

IV. Notes for programmers

As indicated earlier, *Banker* can make both hashed and compressed *G* banks. In the compressed bank, each series has its own starting date and number of observations and most series have been compressed. Compression involves these steps:

1. Find and record the number of decimal points in the series.
2. Slide the decimal to the right until the series is all integers.
3. Record the first observation as a 4-byte integer.
4. Record the first differences of the series as 2-byte integers.
5. If a series cannot be accurately recorded in this compressed form, it is declared to have 255 decimal places (just a flag), and the observations recorded as four-byte floating point numbers.
6. Missing observations are marked with a special code.
7. The data file has the extension "cbk"; and the index, "cin".

The .cin file contains:

```
item                  size in bytes  C type
============================================================
ns                    2              int
nc                    2              unsigned int
names                 nc             char
```

Here ns is the number of series, and nc is the number of characters in all the series names, counting the nulls at the end of each series name. If there were three variables in the bank with names tom, dick, and harry, ns would be 3, and the names vector would be

```
tom0dick0harry0
```

where 0 represents a null ('\0' in C), and nc would be 15, the number of characters in the names vector, counting the nulls.

The .cbk file contains:

item	size in bytes	C type
title	80	char
ns	2	int
position	4	unsigned long
series 1	variable	see below
....		....
series n	variable	see below
indx	4*ns	unsigned long

Here, ns is again the number of series; indx is an array containing the byte numbers in this file at which the series begin. To continue the example, suppose that the series "tom" requires 101 bytes, "dick" requires 121 bytes, and "harry" requires 81. Then "indx" is the vector (86, 187, 308). (Remember that in C a file starts with byte 0). The "position" is the byte number at which this "indx" array begins. In the example, it will be 389.

Each compressed series has the format:

item	size in bytes	C type
BaseYear	1	unsigned char
FreqPeriod	1	unsigned char
SlashDecplaces	1	unsigned char
NDif	2	int
FirstObs	4	long
Differences	2*NDif	int

where

- *BaseYear* = the year of the first observation, minus 1900.
- *FreqPeriod* = 16*frequency+period, where frequency is the number of observations per year (1, 4, or 12)
- *period* = period of first observation. (For frequencies above 12, set FreqPeriod = 255. This value signals that two integers have been inserted following this byte containing the frequency and the period.)

- *SlashDecplaces* = 16*SlashFactor + Number of Decimal places (SlashFactor is normally 0; the Press program, however, allows the option of dividing by a power of 2 to reduce the magnitudes of a series so that it can be compressed. The SlashFactor is the power (1, 2, 3, etc.) used on this series. In Press, the default maximum slash factor is 0, so the occurrence

of non-zero slash factors is unusual.) Usually, SlashDecplaces is just the number of decimal places. If it is 255, the series has not been compressed.

- *NDif* = the number of differences (= Number of observations - 1).
- *FirstObs* = the first observation, as a four-byte integer.
- *Differences* = the first differences of the series, as 2-byte integers.

If a zero occurs in the series, it is indicated by 32767 in the differences. The following difference applies to the previous non-zero number, not to the zero. This practice was adopted because some banks have series with numerous missing observations which appear as zeroes. Also, some banks consider quarterly series to be monthly series in which only the end-of-quarter months have non-zero values.

If it was not possible to compress the series, the format is:

```
item                  size in bytes   C type
============================================================
BaseYear              1               unsigned char
FreqPeriod            1               unsigned char
255                   1               unsigned char
nobs                  2               int
Observations          4*nobs          float
```

Note that the 255 in the third byte is the signal that the series is not compressed. The next two bytes represent the number of observations, and then follow the observations as 4-byte floating point numbers.

The compressed form can represent a series as accurately as can an 18-foot-high graph printed with laser-printer resolution of 300 dots per inch. (All series in the US National Accounts or Industrial Production Indexes compress easily. In general, there may be a certain percentage of series, perhaps as high as 10 percent, that fail to compress.

Hashed banks differ from compressed banks mainly in the organization of their index files. With standard and compressed banks, *G* keeps the names of the series in memory and simply does a linear search for a name each time one is requested. In the hashed banks, the names are grouped into bins on the basis of a number calculated from the letters of the name. When a name is requested, *G* calculates the number, locates the bin in which the name has been put, reads in the names in that bin, and does a linear search over only those names to find the desired series. The size of compressed banks is limited by the requirement that the total number of characters in the names of all series must be less than 64,000. In practice, that limit translates to about six or seven thousand series. Hashed banks, in contrast, can go up to several million series. The effective constraint becomes the size of the user's hard disk. The data file has the extension "hbk"; and the index, "hin".

The precise form of the hashed bank .hin and .hbk files is as follows:

```
The ".hin" file contains:
   item                       size in bytes          C type
   ============================================================
   ns                         4                      long
   nbins                      2                      unsigned
   nsb                        2*nbins                unsigned
   ncharb                     2*nbins                unsigned
   posbin                     4*nbins                unsigned
   binname(0)                 nchar[0]               char
   binposts(0)                4*nnmsb[0]             long
   binname(1)                 nchar[1]               char
   binposts(1)                4*nnmsb[1]             long
   binname(2)                 nchar[2]               char
   binposts(2)                4*nnmsb[2]             long
   .
   .
   binname(nbins-1)           nchar[nbins-1]         char
   binposts(nbins-1)          4*nnmsb[nbins-1]       long
```

Here, *ns* denotes the number of series in the bank. The series are separated into "bins". The number of bins in the bank is denoted by *nbins*. The number of series in each bin is denoted by the array *nsb*. The sum of the number of characters in the names (including each '\0') of the series contained in each bin is denoted by the array *ncharb*. The beginning positions in the ".hin" file of the first bytes of the binname() strings is given by the array *posbin*. The string *binname(i)* denotes the concatination (including the \0's) of all the series names in the i-th bin. Finally, *binposts(i)* denotes the array of beginning positions in the associated ".hbk" of the series in the the i-th bin. Of course, the ordering of the series in the binname() and binposts() arrays must be the same.

Consider an example. Suppose that the 3rd bin contains the series "joe", "dave", and "bill". The string binname(3) would be

```
"joe\0dave\0bill\0"
```

Suppose that the starting positions in the ".hbk" bank for the three series are 40700008, 490987, 3378294. The array *binposts(3)* would then be [40700008, 490987, 3378294]. And *nsb[3]* = 3, and *ncharb[3]* = 14. If the beginning position of *binname(3)* in the ".hin" file is 4724, then *posbin[3]* = 4724.

To assign a bin number to a series you must use the following hashing routine. In C, the routine is:

```
unsigned hash(char *s);

hash(char *s)
{
   unsigned bill;

   for (bill=0;*s!='\0';s++) bill = *s + 31*bill;
   bill = bill%nbins;
```

```
return(bill);
}
```

To continue with the example, to determine the bin which the series "joe" really belongs to you'd evaluate the function hash("joe").

The .hbk file:

```
0 – 79        char        Name of bank (terminated with a null)
80 – 81       int         ns, number of series in the bank
82 – 85       long        psn, position in file of index
86 -          first series, as described below
*(psn+1) -    second series,
...                         ...
psn           long        position in file of first byte of first series
psn+4         long        position in file of first byte of second series
                                 on out to ns series
```

For each series, the format is:

```
byte  Content
0           base year
1           frequency*16+period
2           slash*16+maxplaces or 255 if not compressed
3-4         number of observations
5-8         first observation as a long
9 -         differences as integers
if not compressed, floats begin in byte 5
```