Chapter 16
STORING DATA ON EXTERNAL FILES

The free-format data loading described in Chapter 3 is easy and convenient to use, but it may not be not suitable for large amounts of data: the data may not be in a form that can be read this way; it is somewhat inefficient to use free format; or you may have a large number of series in your data set, only a few of which you wish to use at one time. TSP provides several alternative formats for the storage and use of data in external files.

These alternate file formats are generally used in different situations and for different data problems. In this chapter, we try to give some guidelines for their use. Here are some possible situations:

1. You have a moderate number of not very long series (approximately 50 observations). You could type the data into a free format file, or a spreadsheet file (.WKS, .XLS, etc., but not spreadsheet notebooks).

2. Your project involves a large amount of data: either panel data with several hundred observations on a few time periods for each variable, or a large cross section (more than one thousand observations). Cost considerations dictate that data storage and input/output be as efficient as possible. In this case, the most efficient technique is to store the data in a TSP databank. Presumably the dataset will be read first from a large file.

3. You are building a large model section by section: the total number of series involved is quite large, but they may have different starting dates and frequencies. This kind of project is also well suited to a databank. You can easily store the documentation for each series along with the data values.

The next few sections discuss data storage alternatives in more detail and give a few examples of using them.

16.1. Using external sequential files: READ, WRITE

A *sequential* file is just what its name implies: one record follows the next, in sequence. The file is always written and read in a fixed order.

In TSP, the reading and writing of data to an external sequential file are done with the READ and WRITE commands, using the FORMAT= and FILE= options. The FORMAT= option tells TSP whether the data on the external file is free format, formatted (characters), or binary.

In a *formatted* data file (sometimes called an ASCII file after the character code used), each number occupies a prespecified number of spaces, and is represented by a series of characters in a particular machine code representation. The advantage of formatted data is that it can be printed easily and is readable on different kinds of computers.

Binary or *unformatted* data, on the other hand, is not readable on a different kind of computer than the one on which it was written, nor can it be printed directly; it is the most efficient and the least flexible form of storage. The format of this data is the same as the machine representation of the numbers; so no recoding is done as it is read or written and maximum efficiency is achieved.

For further information on the use of formatted and unformatted data see FORMAT in the *Reference Manual.*

The FILE= option is used to specify the external file from which the data will be read or to which the data will be written.

```
READ(FILE='FOO.DAT') X Y Z;
```

specifies that TSP is to read the series X, Y, and Z in free format from the file FOO.DAT .

16.1.1. Data organization on external files

You can organize data on an external file by series or by observation. Data organized by series typically has one or more records per series, with all the observations, and starts a new record for each series. Data organized by observation is more typical of cross-section research: each record contains all the variables for a single observation and each observation starts with a new record. TSP can read/write data stored in either way. For example, to load data stored by series:

```
FREQ A ; SMPL 48 79 ;
READ(FILE='macro.dat',BYVAR) GNP  IMPT  RELP ;
```

The external file macro.dat has 32 numbers for GNP (possibly on more than one line), then 32 numbers for IMPT, and 32 numbers for RELP.

The next example shows how to read data stored by observation: each pair of records on the input file consists of data on the 11 variables being read; there are 385 pairs of records corresponding to the SMPL of 385 observations:

```
FREQ N ; SMPL 1 385 ;
READ(FILE='PATENTS.DAT',FORMAT='(7F10.5/10X,4F10.5)')
  ID PATENTS LRNDL5 LRNDL4 LRNDL3 LRNDL2 LRNDL1 LRNDL0 TIME BOOK71 SCISECT ;
```

Suppose that you now wish to reformat the data in the second example as a TSP databank file for efficiency in storage and reading. You can use OUT and KEEP statements to do this:

```
OUT PATDATA;
KEEP  ID PATENTS LRNDL0 LRNDL1 LRNDL2 LRNDL3 LRNDL4 LRNDL5 TIME BOOK71 SCISECT PFIT U ;
```

In this example we assume that you also wished to add two computed variables (PFIT and U) to the databank PATDATA. On most computers, the new file takes approximately 20,000 bytes, whereas the old file took about 54,000 bytes, even though it had fewer variables. An easier way to save all original and transformed variables is to put the OUT command before the READ command; then the KEEP command is not needed.

In dealing with large cross-section data files organized by observation, you often do not know the exact number of observations in the file before you make the first run. The READ command has a convenient option to solve this problem, called SETSMPL. SETSMPL does exactly what its name implies: it sets the SMPL after the data is read and the number of observations is known. This option is the default if the SMPL has not yet been defined. For example if PATENTS.DAT was actually a free format file:

```
READ(FILE='PATENTS.DAT')  ID PATENTS LRNDL0 LRNDL1 LRNDL2 LRNDL3 LRNDL4 LRNDL5
  TIME BOOK71 SCISECT ;
```

After these commands have been executed the SMPL will be 1 385. If you are not using FREQ N, provide the frequency and a SMPL with the correct starting date of your file, and then use the SETSMPL option explicitly in the READ statement to update the ending date of the SMPL.

16.1.2. Closing external files: CLOSE

Files are opened when they are first referenced in a READ or WRITE statement and are not closed until the end of the program, unless:

1. The same filename (or unit) is used in both READ and WRITE statements.
2. Many different files (usually more than 12) are used in a single program.

In these situations, the CLOSE command is available to give the user explicit control over the closing of files. CLOSE uses the option FILE=, just like READ and WRITE. Most users will never have to use the CLOSE command, since the default methods are adequate for most file operations.

16.2. Spreadsheet files

TSP can read series and matrices directly to or from spreadsheet files. The following table lists the file types supported by TSP:

Spreadsheet*	Version	Filename Extensions	TSP Support
Lotus 123, Symphony	1,2	.WKS .WK1 .WRK .WR1	Read, Write
Lotus 123	3	.WK3	Read
Lotus 123/J (Japanese)	1,2	.WJ1 .WJ2 .WK2 .WT2	Read, Write
Microsoft Excel	2	.XLS	Read, Write
Microsoft Excel	3,4	.XLS	Read
Microsoft Excel	5	.XLW	Read
Quattro Pro		.WQ1	Read (PC only)

* Note that TSP generally writes the oldest file formats, which are always readable by more recent spreadsheet releases. The exception is Excel where TSP writes in the format of version 2.

Spreadsheet files should take the format of the following example, which consists of quarterly data from 48:1 to 49:1:

	A	B	C
1	Date	CJMTL	PMTL
2	Mar-48	183.4	#N/A
3	Jun-48	185.2	.436
4	Sep-48	192.1	.562
5	Dec-48	193.3	.507
6	Mar-49	206.9	.603

Series are read from individual columns in the file. Series names are optionally supplied in the first row of the file (aligned) above the data columns. Dates may be given in the first column. Many different file configurations are possible, and it is possible to read in some series while ignoring others. Following are some simple guidelines for creating a spreadsheet file for TSP:

1. Put the column names in the first row. They should be valid series names in TSP (lowercase is fine - it will be converted to uppercase, but imbedded blanks and special characters are not allowed). If the file has no names, you can supply them when you read the file in TSP, but this can be inconvenient. If the file contains invalid names, the data can be read by TSP as a matrix, ignoring the current names, and you can supply your own names inside of TSP. TSP will not recognize names in lower rows or in sheets after the first. TSP treats these names as missing values and numbers below them will be read as data for the original column names.

2. The second row must contain data.

3. If you are reading time series, the first column should contain dates. This will ensure the series are read with the proper frequency and starting date, regardless of the current FREQ and SMPL in TSP. Dates can be strings such as 48:1 or numbers formatted as dates (Mar-48, 3/31/48, etc.). You only have to supply enough dates so that TSP can detect the frequency (5 is enough to distinguish between quarterly and monthly). TSP ignores any dates after these, assuming that the data is contiguous (no missing periods/years, or SMPL gaps in TSP terminology). If you have missing periods/years for all series, leave the corresponding rows blank. Below is a table of examples showing recommended ways of defining the starting date and frequency with a dates column:

first date	second date	resulting frequency
(string dates - first character is ' ^ or " in Lotus)		
1		2 A if current FREQ is A, otherwise N
48:	any	A
48:5	any	M
48:4	49:1	Q
1948:1	1948:2	M or Q, depending on further dates
48:1	49:1	A
a2:3	any	invalid
48:2	48:1	invalid
(numeric dates)		
12/31/48	12/31/49	A (any dates 365-366 days apart)
12/31/48	1/31/49	M (any dates 28-31 days apart)
12/31/48	3/31/49	Q (any dates 90-92 days apart)
12/31/48	1/ 1/49	N (any other date range)

If you have dates in other columns, they will be read as numbers. If you are reading a matrix, the date column will be ignored (i.e. it will not be read into the matrix).

4. Missing values can be represented by blank cells or by formulas which evaluate to NA, @NA, #N/A, etc.

16.2.1. Reading spreadsheet files

To read in a spreadsheet file, use the FILE='filename' option. FORMAT=LOTUS or EXCEL is optional. If the filename contains one of the extensions listed earlier (.WKS, .WK3, .XLS, etc.), TSP checks the first few bytes of the file to confirm that it is one of the spreadsheet versions listed above. Conversely, if FORMAT=LOTUS or EXCEL is specified, but the filename does not contain a extension , then .WKS or .XLS is appended to the filename. To read a matrix (bypassing column names and dates), use the TYPE=GEN option. TYPE=CONSTANT is not supported for Lotus files; series will be defined instead. The NCOL=, NROW=, IFULL, UNIT=, etc. options are ignored, but SETSMPL is supported.

If no series names are supplied on the READ command, TSP looks for column names in the file and creates series with those names. If you supply series names, TSP attempts to match them to column names in the file. If the file does not have column names, you must supply a READ argument for each data column. If you are unsure of the file's contents, check it with your spreadsheet or read it as a matrix. If you think the file has column names, but you don't know what they are, try supplying a dummy name that won't be matched and TSP will print an error message listing the column names in the file. If you are reading a matrix (using TYPE=GEN as mentioned above), TSP will create a matrix named @LOTMAT unless you supply an argument (the name of a matrix) to READ.

If for some reason your series are in rows instead of columns, you can read the file as a matrix, transpose it, and UNMAKE the matrix into series.

TSP checks the first row in the file for string names (Lotus cells beginning with the characters ' ^ or "). The names are truncated to 8 characters (if necessary), and are translated to uppercase. They must be aligned above their corresponding data columns. If you have dates in the first column, no name is required for the date column. Any names with imbedded blanks will be ignored.

16.2.1.1. READ examples for spreadsheet files

We will use the SML.WKS file shown below in the following examples:

	^CJMTL	^PMTL
'48:1	183.4	NA
'48:2	185.2	.436
'48:3	192.1	.562
'48:4	193.3	.507
'49:1	206.9	.603

```
READ(FILE='SML.WKS');       ? the series CJMTL and PMTL are defined.  FREQ Q and
                            ? SMPL 48:1, 49:1 are set if there is no current FREQ or SMPL.

READ(FILE='SML.WKS',TYPE=GEN);          ? creates the 5x2 matrix @LOTMAT, with the values
                                        ? of CJMTL and PMTL in its columns.

READ(FILE='SML.WKS') PMTL;          ? only reads in PMTL
```

Here is the nm3.wk1 file (as shown in Lotus, using numeric dates) for the following examples:

04/30/57	23.2	34.5	10.9
05/31/57	23.6	35.1	11.0
	23.9	35.8	11.2
	24.0		11.5

```
READ(FILE='NM3.WK1') SF LA SD;   ? defines the monthly series SF, LA, and SD from 57:4 to 57:7.
                                 ? If there is no current SMPL, this is the new SMPL with FREQ M.
                                 ? The series LA will be given a missing value in its last observation.

READ(FILE='NM3.WK1') SF;         ? An error message is printed because three series names are required
                                 ? unless the option TYPE=GEN is supplied.
```

16.2.2. Writing spreadsheet files

The WRITE command will write either a single matrix or several series to a spreadsheet file. If the file specified already exists, it is overwritten by the new file. If you write a matrix, the file will consist only of numbers. If you write series, their names will be put in the first row. The first column will contain dates (or observation numbers), and each series will be put in a column below its name. Series are written under the control of the current sample. If there are gaps in the sample, observation numbers will be used instead of dates in the first column. Dates are written as the last day of each period, and formatted as Mon-Yr.

16.2.2.1. WRITE examples for spreadsheet files

```
FREQ Q;                                  ? This creates a file equivalent to the READ examples,
SMPL 48:1,49:1;                          ? except dates are formatted numerically instead of as strings
READ CJMTL;  183.4 185.2 192.1 193.3 206.9;
READ PMTL;   . .436 .562 .507 .603;
WRITE(FILE='SML.WKS') CJMTL,PMTL;

FREQ Q;                                  ? This creates a file with the same data columns, but no
SMPL 48:1,49:1;                          ? dates or series names (since a matrix is used).
```

```
LOAD CJMTL;  183.4 185.2 192.1 193.3 206.9;
LOAD PMTL;   .    .436 .562 .507 .603;
MMAKE M CJMTL,PMTL;
WRITE(FILE='SMM.WKS') M;
```

SMM.WKS file:

	A	B
1	183.4	NA
2	185.2	.436
3	192.1	.562
4	193.3	.507
5	206.9	.603

16.3. TSP Databanks

TSP databanks provide a convenient means of automatically storing and retrieving your variables from disk.

The method for storing and retrieving variables from databanks is implicit. Instead of listing variables in a READ statement, the neames of databanks are listed in an IN statement. Any variables in the databanks are then available, and are read in automatically if referenced. The corresponding statement for storing variables is OUT; which marks all variables created or modified after it for output to files listed on the statement. For example,

```
FREQ A;  SMPL 48,87;
IN USNIPA;  OUT USNIPA;
OLSQ GNP C,CONS;                          ? old series read from USNIPA databank
LGNP = LOG(GNP);  LCONS = LOG(CONS);      ? new series saved in USNIPA
```

reads in GNP and CONS from databank USNIPA, runs a regression using them, takes their logs, and add the new logged variables to the same databank.

16.3.1. Storing variables in a databank: OUT

Variables created or modified during a TSP run will be stored as members of a databank at the conclusion of the run if you include a statement of the following form in your TSP run:

```
OUT FILENAME ;
```

After an OUT statement, all TSP variables that were created or modified are written to the file associated with that filename. The variables stored can include series, constants, matrices, or equations, provided they have been created or changed after the appearance of the OUT statement. The example above would create a file called FILENAME.TLB.

A new OUT statement later in the run will cause all items created after its appearance to be put on the file(s) named on it, but will leave the items stored previously on the first file(s) unchanged. The statement

```
OUT ;
```

causes TSP to cease storing anything on the output files. Here is an example:

```
OUT DB1 ;
PRINT W1 LNX1 ; GENR GNP=G/P ;
OUT DB2 ;
GENR W1ACT=W1 ; GENR W1=A1+B11*LNX1 ;
OUT;
GENR W2=A2+B22*LNX2 ;
END ;
```

After this run is completed, the series GNP will be stored in DB1.TLB, W1ACT and W1 will be stored in DB2.TLB, and W2 will be stored on neither file.

Occasionally it is convenient to store variables during a TSP run without having to create or modify them. To do this, use a KEEP statement:

```
KEEP VARNAME1,VARNAME2,.....;
```

This puts the variables named VARNAME1, VARNAME2, etc. in the file referenced by the current OUT statement. This is the only way (other than WRITE(FORMAT=DATABANK,...)...;) to store equations and @ variables in a databank. Standard OUT commands ignore these variables. Only LISTs and PROCs cannot be stored in databanks at all. The KEEP command below stores the equation EQ1 and the variance-covariance matrix @VCOV in the databank FILE1.TLB in addition to the series X and Y and the estimated parameters A and B (which are automatically stored).

```
NAME DBRUN 'TEST OF KEEP COMMAND FOR TSP DATABANK' ;
FREQ A ; SMPL 66 75 ;
OUT FILE1 ;
LOAD X ; 11 9 14 30 18 12 29 35 31 25 ;
LOAD Y ; 1 2 3 4 5 6 7 8 9 10 ;
FRML E1 Y=A+B*X ;
PARAM A B ;
LSQ E1 ;
KEEP E1 @VCOV ;
END ;
```

If you want to save all the series and variables in your TSP program, use the command KEEP ALL.

16.3.2. Documenting variables on a databank: DOC

Since TSP databanks are permanent storage for your data, and may be kept for several years and updated, it is useful to store documentation about your data with the actual series. Otherwise, it is easy to forget exactly what they are if you leave a project for a few months and then return to it. TSP provides the DOC command to let you store information about variables in your databanks. For example, to store descriptions for the preceding example, include these statements somewhere after the OUT FILE; statement:

```
DOC X 'The exogenous variable from the DBRUN example' ;
DOC Y 'The endogenous variable from the DBRUN example' ;
DOC E1 'The equation from the DBRUN example' ;
```

16.3.3. Retrieving variables from a databank: IN

When a TSP databank has been created, it can be accessed by including a statement of the form

```
IN FILE1,FILE2,...;
```

If there is more than one filename in an IN statement, each file will be searched until the relevant variable is found. Up to 8 filenames may appear on one IN statement. When another IN statement is encountered with new filename(s), TSP

will stop searching the first group of files and search the new group for any new items encountered after the second IN statement. The statement

```
IN ;
```

will cause the program to cease searching any files until a new IN statement is encountered.

The following simple job shows the minimum requirements for accessing the databank created by TSP in the last example in the previous section:

```
NAME DBRUN 'IN DATABANK STATEMENT' ;
IN FILE1 ;
PRINT X Y ;
```

Note that it is not necessary to set the FREQ and SMPL before the first reference to the databank; normally TSP will obtain these from the first series in the databank if they are not already specified. Since each series is stored with its own frequency and starting date, series of different frequencies may be mixed in a databank, but this is not recommended.

16.3.4. Databank utilities: DBLIST, DBPRINT, DBCOPY, DBDEL

You may forget what a databank contains, and since it is a binary file, you cannot print it or examine it with a text editor. TSP provides commands to show a databank's contents, to copy a databank, and to delete variables from it.

DBLIST prints the variables in a databank, with their associated lengths. The information is printed in the SHOW command format, which lists starting/ending dates for series, matrix dimensions, and values for scalars.

DBPRINT prints the values for all of the series in a databank, using the current SMPL and FREQ. If you are not sure which SMPL and FREQ to use, check the databank first with the DBLIST command. Of course, standard IN and PRINT commands can be used to list selected variables from a databank.

DBCOPY creates a TSP program file from a databank. This file contains all the commands and data necessary to create the databank on another computer. This can be used to move databanks between different types of computers (such as from PC to mainframe), or as an alternative to DBPRINT for listing the values of non-series variables.

TSP also provides the command DBDEL which can delete one or more variables from a TSP databank. DBDEL can erase variables which were put in the databank by mistake, or which are no longer needed. It can also be used to crudely rename variables in a databank, by first copying the variables new names and then deleting the old names.

16.3.5. Using older databanks in TSP 4.4

Since version 4.2, TSP has contained the capability to store documentation with variables, and store scalars in double precision. This means that databanks older than 4.2 must be converted in order to be used. The conversion is automatic; the old databank will be renamed from BANK.x to BANK.T41 (or BANK.T35 if it was from TSP Version 3.5), and the new databank will be called BANK.TLB. If you are converting Version 3.5 databanks, you have to specify the appropriate FREQ and SMPL before conversion.

16.3.6. Using micro-TSP/EViews databanks in TSP: FETCH, STORE

micro-TSP™ is the econometrics package written and sold by Quantitative Micro Software; it is similar to, but not the same as TSP. **Eviews** is the windows version of **micro-TSP**. **micro-TSP** databanks have a different format from TSP databanks, but TSP can read them.

micro-TSP stores only one series in each of its databanks, using the series name with the extension .DB. The databank

files are not written in binary, and are not very efficient, but they can be edited easily with a text editor and can include comments. TSP has FETCH and STORE commands to access this type of databank (the standard TSP IN and OUT commands will not work with these databanks). These commands should be useful if data is already available in this format, or if this format is preferred over the more efficient standard TSP databank.

For example, to read two series from **micro-TSP** databank files X.DB and Y.DB and run a regression:

```
FETCH X,Y;  OLSQ Y C,X;
```

The equivalent operation, using a standard TSP databank file US.TLB is:

```
IN US;  OLSQ Y C,X;
```

16.4. Saving a work session in a file: SAVE, RESTORE

When using TSP interactively (see Chapters 4 and 17), it is often useful to save all current variables in a file, exit from TSP, and then resume the session later. To do this use the SAVE and RESTORE commands. SAVE creates the binary file TSPSAV.SAV, which contains values for all the current variables (series, matrices, scalars, FRMLs, etc.). RESTORE reads this file and essentially re-creates the session as it was when the user exited from TSP. (The command stream, including PROCs, could also be restored by using INPUT with a renamed BKUP.TSP file).

An alternate name to TSPSAV.SAV can be specified as an argument to SAVE or RESTORE, but only files with the (default) .SAV extension can be used. There is no compatibility with **micro-TSP** SAVE and RESTORE files. SAVE and RESTORE are not intended for permanent storage of data. Databanks are preferred, since they are more flexible. Databanks allow access to individual variables without having to bring all the variables into memory, and can be incrementally updated.