CORONAL DIAGNOSTIC SPECTROMETER
SoHO

CDS SOFTWARE NOTE No. 10

Version 1.1 | 6 June 1994

THE UIT DATABASE SYSTEM

W. B. Landsman
ST Systems Co.

Edited by:
W. Thompson
Applied Research Corporation
NASA Goddard Space Flight Center
Laboratory for Astronomy and Solar Physics
Code 682.1
Greenbelt, MD 20771, USA

William.T.Thompson.1@gsfc.nasa.gov
pal::thompson

Editor's Note: This document was incorporated as a CDS software note from the IDL Astronomy User's Library. Except for the header, the document has not been changed in any way. The reader should be aware that only the software component has been implemented by CDS, and the actual databases referred to—particularly those in the Appendix at the end—do not exist on any CDS computers. Also, some routines not applicable to CDS may not be implemented. The dates and version numbers on the cover page are specific to CDS and do not necessarily reflect any changes made to the document by the UIT team.

One important change that has been made to the software is that DBCREATE now supports an /EXTERNAL keyword, which signifies that the database is to be written in a host-independent external format.

1 INTRODUCTION

This document describes the use of an IDL database system that has been implemented on the UIT computes. The database software was designed by Don Lindler of the GHRS group, and has been adopted by the GHRS and IUE groups. The software is noteworthy for its speed and versatility. An example of its speed is provided by a position search of the 245,000 entries in the IRAS point source catalog, which can be performed in a couple of seconds on a SparcStation 2. The versatility arises because the database software shares the IDL programming, plotting, and image display syntax. The software will run on any computer that has IDL installed, and has been used on Vaxes, Unix workstations, and PC/Windows machines.

In discussing the database software, it is useful to have in mind a book copy of, for example, the Yale Bright Star Catalogue 5th Edition (called 'YALE_BS' in the computer database). The data for a specific star is contained in one row, while column headings are placed at the top of a page. Similar concepts apply to the computer database, and the following terms will be referred to constantly.

entry a "row" of a catalogue. The YALE_BS catalog contains 9110 entries.

item a "column" of a catalogue. The YALE_BS catalog contains 41 items including 'HD', 'NAME', and 'V_MAG'.

value the field corresponding to a specified entry and item. It can be either numeric or a character string. For entry 1708 of the YALE_BS catalog, item 'NAME' contains a value of 'ALP AUR' and item 'V_MAG' contains a value of 0.08. It is possible for an item to be multiple valued. For example, the 'COPERNICUS' database contains *Copernicus* spectra of 40 hot stars. In this case, the value of the item 'FLUX' for a particular entry consists of 2250 numbers specifying the relative flux between 1000 and 1450 Å.

The managers of the IDL database would greatly profit by suggestions from the users. In particular, the managers would appreciate hearing about (1) astronomical catalogues that should be added to the database (2) improvements needed in the help files of a database or items within a database, (3) catalogues that should be linked together via pointers (4) items that should be indexed. (As explained below, indexed items require more disk space, but can be searched much more quickly than non-indexed items.)

Section 2 of this document describes the six "core" database procedures, which may be all that are ever required by the typical user. Section 3 discusses five more special-purpose procedures and the concept of "pointers." Section 4 gives instructions on how to create or modify a database, and need only be read by would-be experts. This document is most effectively read while sitting at a terminal, where the numerous examples can be worked out.

2 DATABASE FUNDAMENTALS

All of the database procedures begin with the letters "DB". As with all other MOUSSE procedures, help on a specific database procedure can be obtained by typing ?<procedure_name>. These help files provide more detailed information than is given in this document. Although there exist a total of 35 database procedures, the following six "core" procedures will often be all that are required by the user.

DBOPEN Open a catalogue for subsequent processing.

DBHELP Display information on catalogues, or items within a catalogue.

DBFIND Find entries meeting specified search criteria.

DBPRINT Print catalogue information of specified entries and items.

DBEXT Extract specified values as vectors for subsequent processing with IDL.

DBCLOSE Close a catalogue.

In addition to the "core" procedures above, there are seven other database procedures that may be used occasionally.

DBCIRCLE Search for entries in positional catalogue within a specified radius of a specified center

DBGET Use instead of DBFIND when search values are in an IDL vector.

DBMATCH Find one entry for each element of a vector of item values.

DBSORT Sort a list of catalogue entries by any item.

DB_OR Remove duplicate values from a list of entries.

IMDBASE Find entries within an image with a specified FITS header

TVDBASE Overlay position of entries within image with a specified FITS header

The rest of the 35 database procedures are either low-level, or used for building databases (see section 4).

2.1 DBOPEN and DBHELP

The database commands discussed here should all be entered in response to the IDL prompt. To list the names of the online catalogues, one simply types DBHELP. Alternatively, the command DBHELP,1 will print a one-line description of all catalogues, and the command DBHELP,<name> will print a description of a specified database. Before any further work can be done with a database, it must be opened with the DBOPEN command. If later, one decides to work with another catalogue, then the DBOPEN command must be used again. DBOPEN will close the first database and open the second. Once a database has been opened, DBHELP is used to give the name, type of data, and brief description of items within the database. (If an item is multiple-valued, then a number in parentheses will appear beside the item name.) The command DBHELP,1 will also show which items are indexed, and thus much quicker to search on. The following example illustrates these ideas:

`$IDL`	!Get into IDL
`DBHELP`	;List the names of all databases
`DBHELP,1`	;Give a one-line description of all databases
`DBHELP,'PTL'`	;Print a brief description of the ASTRO Program Target List (PTL) database
`DBOPEN,'PTL'`	;Open the PTL database
`DBHELP`	;Print the names of all items in the PTL
`DBHELP,1`	;Give the datatype and a brief description of all items and show which ones are indexed
`DBHELP,'JOTFID'`	;Print a description of the JOTFID item
`DBOPEN,'SAO'`	;Close the PTL catalogue and open the SAO catalogue
`DBCLOSE`	;Close all catalogues

The output device of all the database procedures is controlled by the TEXTOUT keyword. The default output device (TEXTOUT=1) is the user's terminal. Set TEXTOUT=3 to direct output to a disk file with a default name, or set TEXTOUT = 'filename', to specify the output file name. The non-standard system variable !TEXTOUT can also be used instead of the TEXTOUT keyword. The documentation for the TEXTOPEN procedure gives a complete description of the TEXTOUT keyword.

For example, if no database has yet been opened, then the following commands would write a one-line description of all catalogues to a disk file.

`DBHELP,1,TEXT = 3`	;Write a one line description of all catalogues to a disk file DBHELP.PRT
`DBHELP,1,T='DB.TXT'`	;Write a one line description to a file DB.TXT

2.2 DBFIND

The function DBFIND is used to select the entries of interest in a catalogue. The general format of the DBFIND call is

```
LIST = DBFIND('SEARCH_CRITERIA',[ SUBLIST ] )
```

LIST is an IDL longword output vector containing the desired entry numbers. It is subsequently used either by DBPRINT to display the desired entry values, or by DBEXT to extract item values for plotting or analysis. SUBLIST is an optional input parameter that restricts the search to a subset of the catalogue. SEARCH_CRITERIA is a string or string array that contains the desired search items. Search criteria can be selected in seven different ways. For example, the JOTFID item (Joint Target File ID) in the PTL database could be searched in the following ways:

	Search Format	Example
(1)	ITEM = value	JOTFID = 8102
(2)	ITEM = [value1,value2]	JOTFID = [8104,8105]
(3)	ITEM = min_value < ITEM < max_value	7000 < JOTFID < 7999
(4)	ITEM > min_value	JOTFID > 3000
(5)	ITEM < max_value	JOTFID < 2999
(6)	ITEM = value(tolerance)	JOTFID = 5000(2000)
(7)	ITEM ;non-zero value	JOTFID

The > and < signs in (3) – (5) are interpreted as less than or equal to (i.e. example (5) would include all jotfid numbers up to and including 2999).

You can use two or more search criteria at the same time by separating the individual criteria with commas. For example, to find the UIT observations of normal galaxies (6000 $\leq$ JOTFID $\leq$ 6999) in the ASTRO target list,

```
dbopen,'PTL'                                 ;Open the Program Target List
list = dbfind('U=U,6000 < jotfid < 6999')    ;Specify UIT target, JOTFID range
```

Suppose, one now one wants to further restrict the list found above to targets in the northern hemisphere. The DBFIND search *could* be repeated adding the additional search criterion `'dec > 0'`. However, it would be quicker to restrict the search for positive declinations to the entries that have already been selected and stored in the vector LIST:

```
newlist = dbfind('dec>0',list)
```

When using DBHELP to display the contents of a catalogue, certain items are identified as being "indexed." Indexed items can be searched *much* faster than non-indexed items. *Use indexed items whenever possible in your search criteria.* (The reason that not every item is indexed is that such items require more disk space.) For example, one could find the star α Aur in the SKYMAP star catalog as follows:

```
dbopen,'skymap'                              ;Open the SKYMAP catalog
list = dbfind('name = alp aur')              ;Slow non-indexed search for the star name
```

but the search will take a couple of minutes since NAME is not an indexed item. On the other hand, a search on the HD number will be almost instantaneous.

```
list = dbfind('hd=34029')                    ;Fast indexed search on the HD number
```

The most commonly used items with DBFIND are probably those pertaining to position. All of the positional databases will have an indexed item named RA in decimal hours, and an indexed item named DEC in decimal degrees. In addition, another set of non-indexed items may exist (e.g. RA_1950, DEC_1950) which store the position as character strings. These latter items are used for pretty output with DBPRINT, *and should not be used with* DBFIND. A nice feature of DBFIND is that it recognizes numbers separated by colons as being in sexigesimal format (e.g. 45:30 = 45.5). For example, to determine if 3C273 is an IRAS source, we could search on the known position with a $15''(= 1^s)$ tolerance:

```
dbopen,'iras_psc'                              ;Open the IRAS point source catalog
list = dbfind('ra = 12:26:33.3(0:0:1),         ;Search on known position of 3C273
   dec=2:19:43(0:0:15)')
dbprint,list,'*'                               ;Print all items in table format
```

The colons can be used with any item, although, of course, the use of sexigesimal format is most common with RA and Dec. The help file for DBFIND can be read to learn how to use slashes and dashes to encode the date and time. However, be aware that arithmetic operations *cannot* be done within the search_criteria string of DBFIND; if the right ascension is 30 degrees, then conversion to hours must be done before the DBFIND call

```
list = dbfind('ra=30/15.')                     ;ILLEGAL statement - do not try this!
```

String searches are matched whenever the supplied string appears anywhere in an item. Thus, in the YALE_BS catalog

```
list = dbfind('name = tau')
```

will find all stars with the three characters 'tau' appearing anywhere in their name. The search is case insensitive and leading and trailing blanks are ignored. One can also use the same item twice in a search. Thus

```
list = dbfind('name = tau,name=eri')
```

will find all stars (nine of them!) with the substrings 'tau' and 'eri' in their name.

2.3 DBPRINT

DBPRINT will display selected fields of database on the device specified by the TEXTOUT keyword. The basic calling sequence is:

```
DBPRINT, LIST, ITEMS, TEXTOUT = , FORMS =
```

where LIST is a vector of entry numbers (e.g. as found with DBFIND), and ITEMS is a list of the desired items to print. One line of output will be generated for each entry printed, and fields will be printed with appropiate headings. (Page overflow will occur if the item list doesn't fit on a single line – 80 characters for a typical terminal and 132 characters for a line printer). The list of items to be used can be specified in six different ways:

```
dbprint,list                                   Display default items
dbprint,list,''                                ;Interactively select items via menu
dbprint,list,'jotfid,id1,id2'                  ;Items are in a single string
```

`dbprint,list,['jotfid',' id1',' id2']`	;Items are in a string array
`dbprint,list,'$FILENAME'`	;Items are in a disk file named FILENAME, (one item per line)
`dbprint,list,indgen(3)+1`	;Use items 1-3, (Item 0 is always the entry number)
`dbprint,list,'*'`	;Select all items and print in table format

An useful variant of the first form of DBPRINT is to pass either an undefined variable or an empty string as the ITEMS parameter to DBPRINT. A full screen menu will appear, and the items to be printed can be selected with the mouse. The items selected will be returned in the ITEMS variables, so that one can skip the menu on subsequent calls to DBPRINT.

`f = ''`	;Define an empty string
`dbprint,list,f`	;Interactively select items via menu, print, and return items list in f
`dbprint,list,f,text=3`	;Print to a disk file with items previously selected

The list of entry numbers can be either the output of DBFIND, or a scalar or vector directly specified by the user. Set LIST $= -1$ to print all entries. Entry numbers begin with 1, so that supplying an entry number of zero may give nonsensical results.

`dbopen,'ptl'`	;Open the PTL database
`items = indgen(9)+1`	;Select first 9 items for printing
`dbprint,132,items`	;Print selected items of entry 132
`dbprint,indgen(50)+1,items`	;Print selected items of first 50 entries
`dbprint,-1,items`	;Print selected item, all entries

2.4 DBEXT

DBEXT allows one to extract item vectors from a database for plotting or subsequent analysis. Its basic calling sequence is

```
DBEXT,LIST,ITEMS,V1,V2.,...V12
```

The parameters LIST and ITEMS have the same meanings as in the DBPRINT command. The outputs V1,V2,...are the IDL variable names to be filled with the values of the specified items. For example, to produce a scatter plot of the right ascension and declination of all targets in the PTL:

`dbopen,'ptl'`	;Open the Program Target List
`list = dbfind('ra<24.')`	;Solar system objects have RA=99.9
`dbext,list,'ra,dec',r,d`	;Extract RA and dec
`plot,r,d,psym=3`	;Plot RA vs. Dec for all Astro targets

Thanks to WUPPE and BBXRT, the observed distibution of targets shows some concentration toward the galactic plane.

For another example, de Lapparent *et al.* (*Ap. J. (Letters)*, **302**, L1) have used the CFA redshift survey to display a "slice of the universe". Figure 1 in their paper (the so-called "dancing man") was obtained by plotting galaxy velocity (distance) versus right ascension. The galaxies were restricted to the declination wedge $26.5^{o} \leq \delta \leq 32.5^{o}$, and also $m_B \leq 15.5$, and $V \leq 15000$ km s^{-1} . The following IDL statements will quickly create a rough version of this plot.

```
dbopen,'REDSHIFT'                                ;Open the CFA Redshift Catalogue
list = dbfind('26.5 <dec< 32.5, b_mag< 15.5,     ;Select galaxies meeting search criteria
   1 <vhelio< 15000')
dbext,list,'ra,vhelio',ra,vhelio                 ;Extract RA and velocity vectors
plot, ra, vhelio, psym=3                         ;Plot projection of galaxies on the RA-Vhelio
                                                    plane
```

In this example, galaxies for which a redshift has not been determined were assigned a velocity of zero. Therefore, it was essential that the search on the VHELIO item had a lower limit of 1 and not 0.

It is possible for an item to contain more than one value for a particular entry. This is often true for databases containing spectra, where the wavelength and flux items will be multiple-valued. For example, to plot a spectrum of η Uma (HR 5191) from the TD-1 spectrophotometric catalog:

```
dbopen,'TD1_SPEC'                         ;Open spectrophotometric catalog
dbhelp,'FLUX'                             ;Read how to create wavelength array
w = [1360. + findgen(60)*20., 2740.]      ;Wavelength array is 1360 Å – 2540 Å at 20
                                             Å resolution plus 2740 Å photometer
list = dbfind('bs_no = 5191')             ;Find entry number for η Uma
dbext,list,'flux',f                       ;Extract 61 element flux vector
plot,w,f                                  ;Plot flux vs. wavelength
```

The TD1_SPEC catalog does not include an item for the wavelength array because it is the same for each entry; instead the help file for the item FLUX tells how to construct the wavelength array in a single IDL statement. Note that if more than one entry number were supplied to DBEXT, then the output flux vector would be 2 dimensional, with the first dimension containing the 61 flux values for a particular entry.

The DBEXT command can be combined with the WHERE function of IDL to perform searches on fields not directly present in the catalogue. As an example, we will use the IRAS point source catalog to search for infrared selected high luminosity galaxy and quasar candidates (AGNs).

Following Low *et al.* (1988) (*Ap. J. (Letters)*, **327**, L41) we set our search criteria to be (1) a 25 to 60 micron flux ratio of $0.25 <$ F25/F60 < 3, (2) a galactic latitude greater than 30^{o} , and (3) no previous identification from a stellar catalog. (As a prerequisite for criterion (1), the sources must have measurable fluxes at 25 and 60 microns.) Search criterion (1) cannot be performed using DBFIND since F25/F60 is not an item in the database. Instead one must extract the F25 and F60 vectors with DBEXT, form the ratio, and use the WHERE function to select the desired range. Similarly, since the catalogue does not include galactic coordinates, one must DBEXT the RA and DEC, and then convert to galactic coordinates. Criterion (3) presents a minor problem because DBFIND does not allow a "not equal to" search criterion; however, one can explicitly search on the catalog identifications (item IDTYPE) that are not stellar (IDTYPE = 2).

```
dbopen,'iras_psc'                         ;Open IRAS point source catalog
```

```
list = dbfind(‘idtype =[0,1,3,4],                ;Not in stellar catalogs, detected at 25 and 60
   25_fqual>2, 60_fqual>2’)                        microns
dbext,list,‘60_flux,25_flux’,f60,f25             ;Extract 25 and 60 micron flux vectors
ratio = f25/f60                                  ;Form 25 to 60 micron flux ratio
list = list(where ( (ratio gt 0.25) and          ;Ratio to select for AGNs
   (ratio lt 3.0) ))
dbext,list,‘ra,dec’,ra,dec                       ;Extract RA and DEC vectors
euler,ra*15.,dec,l,b,1                           ;Convert to galactic coordinates
list = list(where ( (b gt 30) or (b lt -30))     ;Select high galactic latitude objects
   )
dbprint,list,‘name,b_mag,                        ;Print name and IRAS fluxes of selected
   12_flux,25_flux,60_flux,100_flux                sources
```

The above sequence of commands above will run slowly because the 25_fqual, 60_fqual items are not indexed, and because not all the desired search items (e.g. galactic coordinates) are in the database. Should similar searches of the IRAS catalog be required often, then the database manager should include and index the desired items.

3 Advanced Databasing

3.1 DBCIRCLE

DBCIRCLE can be used to search a catalog for all sources within a specified radius of a given position. For example, suppose one wants to determine if any quasars are within the 20′ UIT field of the star AE Aqr (JOTFID number 3226).

```
dbopen,'PTL'                                     ;Open the ASTRO Program Target List
l = dbfind('jotfid=3226')                        ;Get entry number of JOTFID 3226
dbext,l,'ra,dec',ra,dec                          ;Extract the Ra and Dec of this star
dbopen,'quasars'                                 ;Open Hewitt and Burbidge (1989) Quasar
                                                   catalog
list = dbcircle(ra,dec,20,dis)                   ;Find sources within 20′ of given RA and Dec
```

DBCIRCLE will display that 3 entries were found in the quasar catalog, and place the entry values in the vector `list`. The vector `dis` contains the distance (in arc minutes) of each quasar found to the specified field center.

3.2 DB_OR

The DB_OR function concatenates the entries found in two different lists, while removing duplicates. For example, suppose one wants to identify the Astro targets that are *either* WUPPE or HUT targets. The command

```
list = dbfind('H=H,W=W')
```

will identify targets that belong to *both* HUT and WUPPE. To find targets that belong to either instruments, one must perform two searches and concatenate the results.

```
list1 = dbfind('H=H')                    ;Get entry numbers of HUT targets
list2 = dbfind('W=W')                    ;Get entry numbers of WUPPE targets
list = db_or(list1,list2)                ;Combine entry vectors and remove duplicates
```

Of course, IDL allows one to combine the three steps above into a single step:

```
list = db_or( dbfind('H=H'), dbfind('W=W') )
```

3.3 Sorting

Up to this point all results were printed in the order stored in the database (by entry number). The procedure DBSORT will sort an entry list on up to nine sort items. Its calling sequence is

```
SORTLIST = DBSORT(LIST,'item1,item2 ...')
```

where LIST is the input list of entry numbers and SORTLIST is the sorted list. Item1 is the primary sort item, item2 the secondary, and so on. For example, the following statements will produce a printout of all IUE high-dispersion observations of the nuclei of of planetary nebulae, (object class 70) sorted by right ascension:

```
dbopen,'IUE'                             ;Open the IUE catalogue
list = dbfind('obj_class = 70,disp=h')   ;Specify object class, dispersion mode
sortlist = dbsort(list,'ra,image')       ;Primary sort is by RA, secondary by image
                                            number
dbprint,sortlist,                        ;Print selected items
   'object,ra_1950,dec_1950,cam_no,image'
```

3.4 DBGET

Suppose one has a list of five IUE SWP images, and wishes to obtain information about the observational parameters. DBFIND can be used to find the entry numbers

```
list = dbfind('cam_no=3,image=[3427,15191,20227,29992,30022]')
```

The SWP camera is camera number 3, and the individual images are identified. For a larger number of images, however, this use of DBFIND breaks down. It is awkward to write each image value in a string, and, in fact, DBFIND can only parse 10 individual values. What is needed is a function that can search on values in an IDL vector, and this is why DBGET was created.

```
images = [3427,15191,20227,29992,30022]  ;Values are in an IDL vector
list = dbfind('cam_no=3')                ;Restrict search to SWP camera
list = dbget('IMAGE',images,list)        ;Search on "images" vector
```

One limitation of DBGET is that it can only be used with one item at a time. Be aware that the number of entries returned by DBGET might not equal the number of values in the search vector; if, for example, an image number is missing, or appears twice (e.g. as both large and small aperture). The function DBMATCH should be used if a one-to-one correspondence is desired between the elements of the search item vector and the found entries.

3.5 DBMATCH

Suppose one wants to find the Gliese catalog number of every star in the Yale Bright Star catalog. Both these catalogs contain an HD item, so one can extract the HD numbers from the Yale Bright Star Catalog, and then use this vector to search for entries in the Gliese catalog.

```
dbopen,'YALE_BS'                    ;Open the Yale Bright Star Catalog
dbext,-1,'HD_NO',hd                 ;Extract the HD number for all stars
dbopen,'GLIESE'                     ;Open the Gliese Catalog of Nearby Stars
gl = dbmatch('HD_NO',hd)            ;Find Gliese numbers of specified HD numbers
```

The output vector `gl` will contain 9110 elements – one for each entry in the Yale Bright Star catalog. Stars not in the Gliese catalog will contain a value of 0 in the `gl` vector. DBGET could be used to find all the Gliese numbers of stars in the Yale Bright Star catalog, but it would not keep track of which Gliese number went with which star. DBMATCH is slower than either DBFIND or DBGET because it must loop over each element of the item search vector. However, DBMATCH is very useful for building a "pointer" from one catalog to another.

3.6 Pointers

It often happens that the entries in two different catalogues can refer to the same object. It is then possible to open both catalogues simultaneously, and for the entry in catalogue 1 to "point" to the entry in catalogue 2 corresponding to the same object. The user can then print, or search on, items from either catalogue. For example, suppose one wants a printout of comments that have been written about the ASTRO targets, along with the names of the instrument(s) associated with each target. The comments are given in the PTLCOM database, while the instruments are given in the PTL database.

```
dbopen,'PTL,PTLCOM'                     ;Open both the PTL and the PTLCOM
                                            databases
list = dbfind('flag_com')               ;Item is non-zero when comments exist
dbprint,list,'jotfid,id1,h,w,u,b,lcomm' ;Print instrument and comments
```

Use DBHELP to learn if one catalogue points to any others. You *cannot* simultaneously open databases which do not have pointers already built in by the database manager.

4 Creating and Modifying a Database

4.1 Introduction

A database actually consists of four disk files, each identified with a unique 3 letter extension. For example, the PTL database consists of the four files, PTL.DBD, PTL.DBH, PTL.DBF, and PTL.DBX. The .DBD file is an ASCII file that contains all the item definitions, print formats, pointers etc. The .DBH file contains a list of all items and item titles stored in binary format for quick access. The .DBF file contains all the data stored in binary in entry order. Finally, the .DBX file contain the values of all the indexed items stored in binary in item order. The table below summarizes the four database files.

.DBF	DataBase File	Binary	Row-ordered data
.DBH	DataBase Contents	Binary	Title and item descriptions
.DBX	DataBase indeX	Binary	Indexed and sorted data
.DBD	DataBase Definition	ASCII	User-supplied item characteristics

In addition to these four files, the user can create ASCII help files as necessary. For example, the file YALE_BS.HLP will be printed when no database has been opened and the user types `dbhelp,'yale_bs'`. The file YALE_BS_DBLE_NAME.HLP will be printed if the YALE_BS database has been opened, and the user types `DBHELP,'DBLE_NAME'`. A final optional file for a database is an ASCII file with the extension .ITEMS. This file lists the default print items (one per row) to be used when the user types `DBPRINT,LIST`.

The steps a user must follow to create a database are as follows:

- Define the logical name (VMS) or environment variable (UNIX) ZDBASE to point to the directories containing the catalogues. To identify where existing database files are located, type `$sho logical zdbase` (VMS) or `printenv ZDBASE` (Unix).
- Create a database definition .DBD file using a text editor
- Use DBCREATE to create the .DBH file, and empty versions of the .DBX and .DBF files
- Fill the .DBX and .DBF files with data usually using the DBBUILD procedure. Alternatively, entries can be written one at a time into the .DBX file with DBWRT, and the .DBX file can filled using DBINDEX.

4.2 The .DBD file

The critical step in creating a database is making the database description (.DBD) file. Reproduced in full below is a .DBD file for a data base that will be called EXAMPLE.

EXAMPLE.DBD

```
**********************************

#title
Example of a Database catalogue

#maxentries
1356

#items
CAT_NO      I*2     Catalogue Number
BS_NO       I*4     Bright Star Number
RA_1950     C*10    RA (1950)... Use RA for search
DEC_1950    C*9     Dec (1950)... Use Dec for search
```

```
FLUX(61)     R*4      Flux (x 10[-10]), 1380A - 2740A)
RA           R*4      RA (hours)
DEC          R*4      Dec (degrees)

#formats
ENTRY        I6       Entry
CAT_NO       I4       TD1,No
BS_NO        I6       Bright,Star,No
RA_1950      A10      RA,(1950)
DEC_1950     A10      Dec,(1950)

#index
CAT_NO       index
BS_NO        sort/index
RA           sorted
DEC          sort

#pointers
BS_NO        yale_bs
```

****

A .DBD file contains several "block" headers identifiable by a preceding pound sign "#". The #title and #items blocks are required, the #maxentries and #formats blocks are strongly recommended, while the #index and #pointers blocks are optional.

#title Underneath the #title header should be a one line description (50 characters or less) of the database that will be displayed with DBHELP. The actual name of the database (to be used with DBOPEN) is the same as the name of the .DBD file.

#maxentries Underneath the #maxentries header should be a single number giving the maximum number of entries one expects to be in the database. The only cost in making the value of #maxentries too large is that extra disk space will have to be allocated for the index files. On the other hand, if the value of #maxentries is less than the actual number of entries, then you will not be able to create the index file.

#items This required block contains three columns of information. The first column contains the name of every item. Multiple valued items should have the number of values per entry put in parentheses next to the item name. The second column gives the datatype of every item. Acceptable values of datatype include R*4, I*2, I*4, R*8, L*1 or B*1. The datatype of a string item should be written as C*[n] where [n] is the string length. The last column gives a brief description of the item that will be used with DBHELP.

#formats This block lists the item name, print format, and print heading. If an item is not listed here, then it is given the default IDL print format for its datatype (e.g. I7 for I*2 data), and the item name is used for the print heading. Each print heading consist of three rows, so that a heading can consist of up to three words separated by commas. However, each word must fit into the space allocated by the print format; e.g. a heading for the item BS_NO (format I6) will be truncated after 6 characters.

#index This block lists the indexed items and their index types. Multiple valued items cannot be listed here since they are not allowed to be indexed. There are four acceptable index types; "index", "sorted", "sort", and "sort/index". The values of an "index" item are copied in entry order to the index file; this allows the values to be extracted quickly. The values of a "sort" item are copied to the index file in numeric order (i.e sorted), along with a lookup table relating the numeric order to the entry order. "Sort" items can therefore be searched very quickly. 'Sorted' items are assumed to have entry order already coincident with numeric order, and the "sorted" index should be used whenever it is allowed. For example, some catalogs are listed by increasing right ascension, which would allow RA to be a "sorted" item. String items are not allowed an index type of "sort" or "sorted". Finally, the values of a "sort/index" item are copied in both entry order and numeric order to the index file. This type is mainly used for items which are used to "point" at another database (see below). The following table summarizes the relative disk space and search speed of the different index types.

Index Type	Disk Space	Search Speed	Extraction Speed	Comment
None	1	Slow	Slow	
Index	2	Moderate	Fast	
Sorted	3	Very Fast	Fast	Item must be in entry order
Sort	4	Fast	Slow	String items not allowed
Sort/Index	5	Fast	Fast	String items not allowed

#pointers The #pointers block contains the names of items that point to the entry numbers of another catalogue. In our example, BS_NO points to the entry number in the Yale Bright Star catalogue. Item that serve as pointers must be either index type "index", "sort/index" or the item ENTRY.

4.3 Adding or Modifying Data

Once a database description (.DBD) file has been created, adding and modifying data is relatively easy. The procedure DBCREATE is used to create the contents (.DBH) file, and optionally, new copies of the data (.DBF) and index (.DBX) file. Its basic calling sequence is

```
DBCREATE,'<DATABASE_NAME>',[NEWINDEX,NEWDB]
```

where NEWINDEX is non-zero to create a new .DBX file, and NEWDB is non-zero to create a new .DBF file. DBCREATE requires that the user set the system variable !PRIV=2; this is to prevent novice users from accidently corrupting the database.

It assumed that the user has been able to read the data into IDL vectors. (The procedures READCOL and READFMT are extremely useful for reading raw data from ASCII files into IDL vectors.) For example, to create the EXAMPLE database from the previous section, the user should have vectors named, say, CAT,BS,RA_1950,DEC_1950,FLUX,RA, and DEC corresponding to each of the items. The database must then be opened for update by adding a second parameter to the DBOPEN command. It is also necessary for the user to have sole access to the database; an error message will result if one tries to update a database while another user is reading it. Finally, the database is loaded with the procedure DBBUILD as follows:

```
!PRIV=2                                          ;Set !PRIV to create or modify database files
dbcreate,'EXAMPLE',1,1                           ;Need new index and data files
dbopen,'EXAMPLE',1                               ;Open the database for update
dbbuild,cat,bs,ra_1950,dec_1950,flux, ra,dec     ;Load IDL vectors into database
```

To modify the item titles or print formats in an existing database, one simply edits the .DBD file with the new information, and then types DBCREATE. There is no need to create new .DBX or .DBF files. If, however, one wants to change the index type of an item or set of items, then a new .DBX file must be created and built with DBINDEX.

```
!PRIV=2                                          ;As always
dbcreate,'EXAMPLE',1                             ;Create a new index file
dbopen,'EXAMPLE',1                               ;Open database for update
dbindex                                          ;Make the index file
```

There are several ways to append or modify the actual data in a database. To append new entries, the DBBUILD command can again be used, but without first calling DBCREATE, since the data in the existing .DBX and .DBF files must remain. The procedure DBUPDATE can be used to load new item values into a database. DBUPDATE can be viewed as the inverse of DBEXT – instead of extracting item vectors, it will insert them. For example, suppose the RA and DEC items are in 1900 equinox, and one wants to convert them to 1950 equinox.

```
!PRIV=2 & dbopen,'EXAMPLE',1                     ;Ultimately, will update database
dbext,-1,'RA,DEC',ra,dec                         ;Extract RA and Dec vectors
ra = ra*15.                                      ;Convert to degrees
precess,ra,dec,1900,1950                         ;Convert to 1950 equinox
dbupdate,-1,'RA,DEC',ra/15.,dec                  ;Load new values of RA and Dec
```

Finally, the procedure DBEDIT is useful for editing individual item values. For example, suppose a database has values of V_MAG set to 99.9 whenever the visual magnitude was unknown. Once these values become known they can be inserted by hand into the database:

```
!PRIV=2 & dbopen,'EXAMPLE',1                     ;Open for update
list = dbfind('V_MAG=99.9')                      ;Get entry numbers with bad V_MAG
dbedit,list,'V_MAG'                              Interactive editing of selected entries
```

DBEDIT will display the existing value of an item, and prompt the user whether to keep or replace it.

5 APPENDIX: ONLINE CATALOGS

This appendix lists some of the more important on-line catalogues. Use DBHELP to obtain a more complete and up-to-date listing.

ASTRO/UIT Catalogues	# of Entries	Name
Galactic Globular Cluster Database	121	GLOB
ASTRO IPS Actual Science Observations	314	IPSREAL
ASTRO May 9 Joint Science Plan	626	JSCIPLAN
ASTRO May 9 Mission Target List	718	MTL
ASTRO Program Target List	1,184	PTL
ASTRO Target List Comments File	334	PTLCOM
2nd Reference Catalog of Galaxies	4,364	RC2
ST Guidestars in UIT fields	41,948	GUIDESTAR
Virgo Cluster Multibandpass Database	228	VIRGO_UIT
UIT Filter Sequence Database	89	UITSEQ

General Catalogues and Databases	# of Entries	Name
ANS UV Catalogue	3,573	ANS
Copernicus far-UV spectra	40	COPERNICUS
Dorman Horizontal Branch Models (1992)	51	DORMAN
EUVE Bright Source List	356	EUVE
Gliese Catalog of Nearby Stars Preliminary 3rd Edition (1991)	3,802	GLIESE
Catalogue of HST observations	24,239	HST_CATALOG
IRAS Small Scale Stucture Catalog	16,740	IRAS_SSS
IRAS Point Source Catalogue	255,578	IRAS_PSC
IUE Merged Log Catalogue	95,944	IUE
Kurucz Model Atmospheres (1992)	6,455	KURUCZ
Library of Stellar Spectra (Jacoby 1984)	161	LSS
Brightest Stars in M31 field (1988)	20,216	M31STARS
OB stars in M33 (Wilson 1991)	3,005	M33STARS
New General Catalog (NGC) objects (1988)	13,226	NGC2000
Catalogue of Principal Galaxies (Paturel et al. 1989)	73,197	PRIN_GAL
Quasars and AGN 5^{th} ed. Veron-Cetty and Veron (1991)	8,000	QUASAR_AGN5
Catalog of Quasars and BL Lac Objects Hewitt and Burbidge (1989)	4,383	QUASARS
CFA Redshift Survey	31,224	REDSHIFT
ROSAT Wide Field Camera Bright Source List	384	ROSAT_WFC
SAO/CFA Star Catalogue	255,988	SAO
Seyfert Galaxies (Weedman 1977)	121	SEYFERT
Stellar Evolution Sequences Schaller et al. (1992)	50	MAEDER
SKYMAP V3.5 (1989)	248,563	SKYMAP
Sweigart (1987) Horizontal Branch Models	120	SWEIGART
TD-1 UV Photometry Catalogue	31,114	TD1
TD-1 Spectrophotometry Catalogue	1,356	TD1_SPEC
Ultraviolet Spectral Synthesis Library (Fanelli *et al.* 1987)	15	UV_SPEC_LIB
Virgo Cluster Catalog (Binggeli 1987)	2,096	VIRGO
Catalogue of White Dwarfs McCook and Sion (1987)	1,282	WDWARF
IUE spectra of white dwarfs (Wegner & Swanson 1991)	184	WDWARF_IUE
Yale Bright Star Catalogue Preliminary 5th Edition (1991)	9,110	YALE_BS
Yale Bright Star (Remarks)	8,300	YALE_BS_RMKS