Yohkoh Analysis Guide
SSW Reference Manual

Ver 1.06 Ed: M.D. Morrison 6-Nov-92
Ver 2.00 Ed: M.D. Morrison 25-Jan-94
Ver 2.91 Ed: R.D. Bentley 1-Mar-98

This Copy Produced: March 6, 1998

Prepared at
Mullard Space Science Laboratory
University College London

Contents

List of Figures

1 Getting Help on Software and Data

1.1 DOC_LIBRARY [IDL-LIB]

IDL has a facility called DOC_LIBRARY which you can use to get the details of any procedure that you know the name of by typing:

IDL> **doc_library,'yodat'**

This will obtain a listing of the header information of the program YODAT (any comments put in at the beginning of the code between ';+' and ';-') provided it is in your path. Wild cards are allowable, e.g.,

IDL> **doc_library,'test*'**

DOC_LIBRARY is a script driven program (i.e., it will work on ordinary terminals as well as X-terminals or workstations. Do not include the '.PRO' in the call to DOC_LIBRARY.

1.2 FASTDOC

This procedure will print out the IDL documentation header for the input routine(s). The location of the IDL routines are determined using a table lookup algorithm so that subsequent calls are very fast. For routines located within the Yohkoh tree, FASTDOC is an efficient substitute for DOC_LIBRARY, which must search for routines on every call. The */summ* switch may be used to print only the procedure or function definition line. The */multi* switch will print all online versions of the input routines. When called without parameters, an interactive version is invoked. Currently only works on Unix machines.

IDL> **fastdoc, 'fastdoc'**
IDL> **fastdoc, 'fastdoc', /multi, /summ**
IDL> **fastdoc, '*anytim*'**
IDL> **fastdoc**

1.2.1 PR_PATH_LIB

PR_PATH_LIB will search directories in the IDL !path variable in the order of the list to find a given IDL program. The default is to only give the first instance that the file is found, but it is possible to get all locations by using the */multi* switch. A wild card ('*') effectively sets the */multi* switch). For example,

IDL> **pr_path_lib,'gt_*'**

will return a list of all "gt_" routines and their directory locations.

1.3 XDOC

XDOC is a widget version of DOC_LIBRARY. It is invoked by typing:

IDL> **xdoc,'name'**

where NAME is the name of the procedure to be examined. The text of the input procedure will appear in a scroll window, with buttons to control print the file, and also extracting and copying the file to the current directory as .TXT file. Calling XDOC without a procedure name will produce a menu listing of all the directories and/or libraries that are defined in the current IDL_PATH. The user can subsequently scan the IDL path for any desired procedure by clicking on a particular path element.

1.4 CHKARG

CHKARG is a routine that solves the problem of what to enter when you have forgotten the argument list or argument ordering of a particular procedure or function. It is invoked by typing:

IDL> **chkarg,'name'**

where NAME is the name of the procedure in question. CHKARG will echo the calling sequence (arguments and keywords) and also the directory/library location of the procedure.

1.5 XDL [IDL-LIB]

XDL is a widget driven version of DOC_LIBRARY and will present you with a complete list of all the IDL software in your path. You can search using the mouse and slider on the screen. It is run by typing:

IDL> **xdl**

1.6 PRO_LIST

PRO_LIST returns a list of the IDL programs, functions, and procedures (.PRO files) that are in the IDL path (!path).

IDL> **pro_list**

IDL> **pro_list, outfil='pro_list.txt'**

1.7 HELP [IDL]

The IDL command HELP is available to get information on all the variables, procedure and functions currently in IDL's memory. By typing:

IDL> **help**
IDL> **help, data**

you get a list of all of the variables that are defined within the current IDL session. If you want information the definition of a structure, you can use the /STRUCTURE option. Some examples are:

IDL> **help, /st, roadmap**
IDL> **help, /st, index.gen**

It is possible to get information on the calling parameters of procedures and functions that have already been compiled by typing:

IDL> **help, /routine**

The procedure and function name is listed first. The positional parameters are listed in lower case and the keyword parameters are in upper case.

1.8 HELP_PREFIX

HELP_PREFIX will display a list of the Yohkoh three letter prefixes which are used for the data files. The */full* option will show a paragraph about the prefix. Some sample calls are:

IDL> **help_prefix**
IDL> **help_prefix, 's*', outfil='junk.txt'**
IDL> **help_prefix, 'att', /full**

1.9 HELP_FIGS

HELP_FIGS will display a menu of several figures which appear in the Yohkoh guides (one option is to call HELP_ROLL which shows the S/C roll sign conventions). A sample call is:

IDL> **help_figs**

2 General Software

2.1 Routines for Reading and Saving Data

2.1.1 SAVEGEN and RESTGEN

SAVEGEN is a routine available to save variables of any type (including structures). Some sample calls are shown below. If no file name is specified, it will use 'save.genx'.

IDL> **savegen, var1, var2**
IDL> **savegen, var1, var2, var3, var4, var5, file='flare27oct92'**

Use RESTGEN to restore the data. NOTE: It is recommended to use SAV_SDA, SAV_BDA, SAV_BSC, and SAV_HXI whenever possible for standard SXT, BCS, and HXI data sets.

2.1.2 TIFF_WRITE [IDL-LIB]

To write some image data (*image*) to a file (*outfile*) in TIFF format all you have to do is type the following for grey scale:

IDL> **tiff_write, outfil, image**

For 256 element (pseudo) color, use:

IDL> **tiff_write, outfil, image, red=r, green=g, blue=b**

where the optional parameters red, green and blue are used to save the color table characterized by the byte arrays (r, g, b - see TVLCT for details about obtaining them).

To make full (24 bit) color TIFF format files see the documentation for TIFF_WRITE (i.e. use the DOC_LIBRARY utility on TIFF_WRITE).

2.1.3 MK_TIFFB

To make full (24 bit) color TIFF format files to be read on Macintosh computers use:

IDL> **mk_tiffb, outfile, image, r, g, b**

where the required color table is specified by 256 element byte arrays *r, g, b*. The output TIFF file contains the specified image which has been flipped vertically for direct display on (most) tiff-readers on the Macintosh and has a resolution of 100 pixels per inch. The default output file size is 774 Kbytes which results from rebinning into image to 508×508 and fits on the standard double sided Mac floppies (779 Kbytes capacity). For more options (such as changing image size which is 508×508 by default, noflip, etc.) or information on MK_TIFFB use the DOC_LIBRARY utility.

2.1.4 WRT_FITS [*]

WRT_FITS creates a FITS file from an image (*data*) by typing:

IDL> **wrt_fits, outfil, header, data(*,*,n)**

where *outfil* is the string containing the filename. *header* is optional input which is the FITS string array ASCII header. If it is undefined, then WRT_FITS will build the minimal header. The routine RFITS should be used to read it. For SXT data, we recommend that you use SXT2FITS which is described in the next section.

2.1.5 RFITS

RFITS reads a standard format FITS file by typing:

IDL> **img = rfits(infil, header=header)**

where *img* is the image, *infil* is the input file name, and *head* is the string array with the data header in it. It is possible to use the $/SCALE$ option to convert the data into some physical units. For example, the magnetogram data would be converted to gauss.

IDL> **img = rfits('/yd0/gbo92_37/gkm920907.1442', head=head, /scale)**

2.1.6 SAVE [IDL]

SAVE is an IDL facility that will save variables, etc. into a file of your choice. To save a variables *a*, *b*, and *c* to the default file 'idlsave.dat', type:

IDL> **save, a, b, c**

To save to the file 'junk.save', type:

IDL> **save, a, b, c, filename='junk.save'**

To save all variables, type:

IDL> **save, /all, filename='junk.save'**

CAUTION: When restoring the data, the variable names are restored exactly as they were saved, so if you saved *a, b* and *c*, then when you restore that save set, any existing variables *a, b* and *c* are deleted when the restore occurs. A sample command to restore data is:

IDL> **restore, 'junk.save'**

2.1.7 SAVESYS and RESTSYS

SAVESYS and RESTSYS provide a simple method for saving and restoring the state of IDL system variables. Use of this routine pair will help programmers avoid global side effects within routines which change system variables. In general, the use of these routines should be done symmetrically within a given routine as the following example dummy code segment illustrates.

```
pro junk,a,b,c          ; function or procedure definition
savesys,/aplot           ; *** save plot variables with savesys.pro
<change !x,!y,!z,!p>     ; code which changes IDL system variables
restsys,/aplot           ; *** restore plot variables with restsys.pro
return
end
```

Another use of restsys.pro is to restore IDL system variables to the startup state to recover from routines which corrupted the system variables.

IDL> **restsys, /init, /all** ; restore IDL system variables to startup state

2.2 Concatenation of Datasets

2.2.1 STR_CONCAT_TAGS

It is possible to concatenate two structures of same or similar type by using STR_CONCAT_TAGS. For example, if you wanted to concatenate the variable *index1* and *index2*, the command would be:

IDL> **index_out = str_concat_tags(index1, index2)**

2.2.2 STR_DIFF

STR_DIFF will is a boolean function which returns true (1) when differences are found between the two input structures. If no differences are found, it returns false (0).

IDL> **qdiff = str_diff(index1, index2)**
IDL> **qdiff = str_diff(index1, index2, /diff**
IDL> **qdiff = str_diff(index1, index2, dtnames=dtnames**

where */diff* will only check the tag names. *tnames = tnames* is an optional input list of the tag names which should be checked (the default is to check all tags). You can get a list of the tags whose values differ by using the optional keyword output, *dtnames = dtnames*. STR_DIFF is not recursive.

2.2.3 Concatenating Data Arrays

A sample command for concatenating two 3-D arrays is:

IDL> **out = [[[in1]],[[in2]]]**

If *in1* is 100×200×4 and *in2* is 100×200×3, then *out* would be 100×200×7. The first two dimensions of *in1* and *in2* must be identical. It is possible to have *in2* be 100×200, in which case *out* would be 100×200×5.

A sample command for concatenating two 2-D arrays is:

IDL> **out = [[in1],[in2]]**

If *in1* was 100×10 and *in2* was 100×20, then *out* would be 100×30. In this case, the first dimension of *in1* and *in2* must match.

2.3 Routines for Data and Information Extraction

2.3.1 The Concept Behind the GT Routines

A series of GT routines was created so that a single piece of information can be extracted from a structure, whether the structure was a 'roadmap' or an 'index' (the data are saved in a different location). The routines also allow for conversions to a string mnemonic or to physical units. It is possible to get a list of what the different values are by typing a command like:

IDL> **print, gt_filta()**

It is also possible to convert the output value to the string mnemonic by using the */STRING* switch. For example:

IDL> **print, gt_filta(roadmap, /string)**

This command only works for the routines which return a coded value.

2.3.2 WDEFROI [*]

The WDEFROI function allows the user to select an arbitrarily shaped region-of-interest (ROI) from a displayed image via the mouse. The routine returns either a sub-image or a list of indices to the image displayed which defines this sub-image. If a data cube is passed the routine returns either a sub-image data cube or 2-dimensional list of indices. NOTE, WDEFROI is not SXT specific and thus will also work for ground base images.

Some example calls are:

IDL> **list = wdefroi(data, zoom=8)**
IDL> **sub_image = wdefroi(data, zoom=8, /image)**
IDL> **list = wdefroi(data, zoom=zoom, /lc, time=time, sub=sub, /logscale)**

where,

INPUT:

data	Data array for the input image. If image is a data cube: one needs to specify which image (see the *sub* parameter) to display for selecting an ROI.

OPTIONAL INPUT:

zoom	is the zoom factor used on the displayed image.
image	if set will return selected sub-image.
lc	if set and a data cube is given, this will allow user to plot light curves of the ROIs. NOTE: if you pass normalized data these light curves will be more meaningful.
time	time of each image in data cube. This can also be a Yohkoh index data record (e.g. SXT index-data record).
sub	is the image index number to use for ROI selection if the input data are a data cube. Note: if you do not specify a *sub* while using a data cube the routine starts STEPPER to allow the user to select an image on-the-fly.
logscale	if set, will log scale the (decompressed) data for display.

OUTPUT:

sub_img	is the returned SUB-image(s) of the selected region (when the */image* switch is used).
list	is the returned list of selected indices of the input image(s) (default return).

(A) PFI Example: An example of selecting a feature from a 64×64 pfi image: Follow the instructions displayed on the widget.

IDL> **nimage = wdefroi(data, /image, zoom=8)**

(B) FFI Example An example of selecting a small feature from a 512x512 image: Due to the fact that contouring a large image is a very slow process do not attempt to contour a 512 × 512 image directly but rather use the “Box” and “select” features to first isolate the area of interest before using the “contour” feature. NOTE: the “contour” feature can still be somewhat slow after isolating a part of FFI. Follow the instructions displayed on the widget.

IDL> **sub_img = wdefroi(img,/image) ;default zoom is 1**

(C) PFI Data Cube Example: An example of selecting a feature from a cube aligned PFI data set which has the dimensions of 65x65 by 40 images. The first line makes a buffer for the normalized data. The second line puts the exposure normalized data into *dat*. The last line uses the "light curve" option and passes the index information for the image times.

IDL> **dat = fltarr(65,65,40)**
IDL> **for i=0,39 do dat(0,0,i) = data(*,*,i)/gt_expdur(index(i))**
IDL> **sub_img = wdefroi(dat,zoom=7,/image,time=index,/lc)**

The function starts the STEPPER routine to allow user to select image. Once the user "quits" STEPPER the WDEFROI widget will appear. Select a "box" ROI and use the mouse make the selection (use the right mouse button to finalize the selection). Use the light curve button to display a time profile of the total intensity from the selected ROI. Select "Exit" to finalize your selection. Then run STEPPER on the newly made SUB-images.

2.3.3 CONROI [*]

Define a contoured region-of-interest of an image using the image display system and the cursor/mouse. This function returns a vector containing the subscripts of pixels inside the contoured region. Note that these are 'linear' subscripts, not 'X, Y' pairs.

IDL> **result = conroi(data(*,*,n))**
IDL> **result = conroi(data(*,*,n), zoom=4)**

2.3.4 PR_IMAGE

PR_IMAGE is used to get a print out and/or the average count rate in a given area on an image (*data(*,*,n)*) by typing:

IDL> **pr_image, data(*,*,n)**
IDL> **pr_image, data(*,*,n), size**

where *size* is the size of the box to be considered (default is 9×9). It is a mouse driven program.

2.4 Routines for Making Time Plots

2.4.1 UTPLOT

UTPLOT plots any quantity on a time plot with hours, minutes, and seconds given a corresponding index array by typing:

IDL> **utplot, roadmap, gt_total_cnts(roadmap, 1)**
IDL> **utplot, index, timehist**
IDL> **utplot, x, y, ref_time, xrange=xrange**
IDL> **utplot, x, y, ref_time, timerange=[index(0), index(100)]**
IDL> **utplot, x, y, ref_time, timerange=['1-jun-91','5-jun-91']**
IDL> **utplot, x, y, ref_time**

Most typical IDL plotting parameters also apply. You can overplot other quantities (timehist2, etc.) by using OUTPLOT.

2.4.2 OUTPLOT

OUTPLOT works just like UTPLOT but does not refresh the screen so one can plot several different quantities on the same plot (see UTPLOT first). An example is:

IDL> **outplot, index, timehist2**

2.4.3 EUTPLOT

EUTPLOT works like UTPLOT but it over plots vertical error bars on a plot that was previously made with UTPLOT. Example calls are:

IDL> **eutplot,x,y,unc,/unc** ; unc = +/- uncertainties
IDL> **eutplot,x,low,high**

where `low = y - unc` and `high = y - unc`.

2.4.4 PLOT_LCUR

PLOT_LCUR is a low level generic plotting routine used by several other routines. The routine returns the list of subscripts which are between the times selected by the user. The user clicks on the plot with the left button to define a new start time, the right button to define an end time, and the middle button to exit. The user can expand a time range after marking the start/stop times by clicking with the left button on the box in the lower right corner. Some of the more simple calls are:

IDL> **ss = plot_lcur(roadmap, gt_sxs1(roadmap))**
IDL> **ss = plot_lcur(index, gt_sum_l(index), plotr=a, title=title)**

IDL> **ss = plot_lcur(index, gt_sum_l(index), /nohard)**

2.4.5 LCUR_IMAGE

The procedure LCUR_IMAGE will allow a user to select a portion of an image and will display a normalized light curve plot. The region can be a simple rectangle or a polygon. LCUR_IMAGE can be used on data other than SXT images as long as the *index* input has .TIME and .DAY fields (see ANYTIM2INTS to build a structure). Some sample calling sequences are:

IDL> **lcur_image, index, data**
IDL> **lcur_image, index, data, lcur**
IDL> **lcur_image, index, data, lcur, uncert, satpix**
IDL> **lcur_image, index, data, xsiz=512**
IDL> **lcur_image, index, data, xsiz=512, /poly**
IDL> **lcur_image, index, data, xsiz=512, /nonorm, subscript=subscript**

The data are binned and exposure normalized. It is possible to mark multiple locations and to overlay the light curves. The keyword option */plot6* will display the multiple plots in six different sub-windows instead of overlaying the light curves. The regions being marked are displayed and labeled so that light curves for a given region can be matched up. There are hardcopy options. If the user passes in the uncertainty (*uncert)* and the saturation map array *satpix*, it will mark the points which have saturated pixels or large uncertainties. The *xsiz* option will blow up the image to be 512 pixel wide (and 512 pixels tall if it is square) so that the selection of a region is simple. There are several other options which are described in the program header.

2.4.6 TPROFILES

TPROFILES will plot the light curve for any pixel in a data cube (data(x,y,t)). As you drag the cursor around the image, it updates a light curve plot automatically. It differs from LCUR_IMAGE since it updates as the cursor moves (not integrating a box), and plots the intensity versus image number, not versus date and time. To use it just type:

IDL> **tprofiles,data**

The routine is mouse driven and will issue instructions for you.

2.5 Routines for Image Display and Enhancement

2.5.1 STEPPER

STEPPER takes a data cube (*data*) and allows a user to step through the images and to display them as a movie. If the *index* is passed in to stepper (as shown in 2 of the examples below) it is possible to chose an option to display light curves. Some sample calls are:

IDL> **stepper, data**
IDL> **stepper, data, info_array**
IDL> **stepper, index, data, xsiz=512**
IDL> **stepper, index, data, info_array**

where *info_array* and *xsiz* are optional parameters. *xsiz* indicates the size of the displayed image you want (must be an integer multiple of the the original array). It is a window driven program.

The use of the STEPPER routine is described in a little more detail in the User's Guide.

2.5.2 XSTEPPER

XSTEPPER is like stepper but is a widget based program, to run it type:

IDL> **xstepper, data, info_array**
IDL> **xstepper, data**

It is a widget driven program with self explanatory instructions. Some options include the ability to zoom in on a particular region and the ability to call XLOADCT from within XSTEPPER.

2.5.3 XMOVIE

It is possible to load a data cube into a widget to be "animated" using the routine XMOVIE (which is simply an interface for IDL's XINTERANIMATE). The speed which the frames can be displayed is quite fast. There are some memory restrictions which do not allow large data cubes. Some sample calls are:

IDL> **xmovie, data**
IDL> **xmovie, data, ss=ss**
IDL> **xmovie, data, 4**

The second positional parameter is optional and is the rebinning factor. This is quite useful when looking at small 64×64 PFI images.

2.5.4 OCONTOUR

OCONTOUR allows you to plot a contour on top of an image in the same way as you use CONTOUR.

IDL> **ocontour,image**

2.5.5 UP2, UP4, UP8 [*]

To re-bin and display an image, you can use the UPn routines. UP2 will rebin the image to be twice as big and then call TVSCL. UP4 and UP8 increase the size of the image four and eight times respectively. A sample call would be:

IDL> **up8, data(*,*,200)**

2.5.6 RM_DARKLIMB [*]

In order to subtract the limb darkening from Hα or white light pictures you can call:

IDL> **img_out = rm_darklimb(Haimage)**

where *Haimage* is the name of the Hα image with limb darkening and *img_out* is the new Hα image with limb darkening removed. RM_DARKLIMB does not work if the image is oblate. The image should then be viewed with TVSCL (not TV).

2.5.7 GBO_SCALE2 [*]

GBO_SCALE2 can be used for aligning SXT images with each other or with other images. GBO_SCALE2 takes two images, resizes them and lays them on top of one another by typing:

IDL> **outimg=GBO_SCALE2(trgimg, gboimg)**
IDL> **outimg=GBO_SCALE2(sxtimg, gboimg, index1, sxt=1)**
IDL> **outimg=GBO_SCALE2(sxtimg1, sxtimg2, index1, index2, sxt=2, /mktv)**

which will scale the input image *gboimg* and center it to match a target image *trgimg*. If either of these is an SXT image then it needs to have its index passed as well, and the SXT keyword should be set to 1 or 2 appropriately, (i.e., for one or two SXT images).

GBO_SCALE2 corrects for different vertical and horizontal diameters. The MKTV keyword allows you to plot the result directly. This should work with general sized images.

2.5.8 COMPST [*]

There is a routine called COMPST that interleaves two images and scales them to each use half the color table. It does no alignment and requires that you have or develop appropriate color tables. The calling sequence is

IDL> **imageout= compst(image1, image2)**

It is currently set up to work with 512x512 images. It is a good idea to massage the images so that they emphasize the intensity ranges of interest, e.g., for low-intensity features, a decompressed image gives poor results. Creative color tables may also be useful.

2.6 Standard IDL Image Display Routines

2.6.1 TV [IDL]

The command TV takes a 2-dimensional variable and displays it directly to the display (you must have x-windows or a image display device). There is no scaling performed on the data. If the data are not byte type, and there are values over 255, then wrap-around will occur. It is probably best to use TVSCL for these datasets. To display the first image in the variable *data*, type:

IDL> **tv, data(*,*,0)**

To display the "i"th dataset, type

IDL> **tv, data(*,*,i)**

2.6.2 TVSCL [IDL]

TVSCL will first perform a linear byte scaling, making the smallest value 0 intensity and the largest value 255. To display image number 11 in the variable *data*, type:

IDL> **tvscl, data(*,*,11)**

2.6.3 CONTOUR [IDL]

You can contour any given image by typing:

IDL> **contour,image**
IDL> **contour, data(*,*,0)**
IDL> **contour, image, levels=[l1,l2,l3,...,ln]**

where *levels* defines the contour levels you want to plot. *levels* is an optional parameter which IDL will choose for you if you omit them. You can overplot a contour on an image by using OCONTOUR in the same way.

2.6.4 XLOADCT [IDL-LIB]

XLOADCT is a widget driven IDL facility that allows you to change color tables and manipulate them. To run it type

IDL> **xloadct**

2.6.5 LOADCT [IDL-LIB]

LOADCT will load one of the multiple standard IDL color tables. To load the red color table, type:

IDL> **loadct, 3**

2.6.6 TVLCT [IDL]

TVLCT allows a user to load a color table which he has already defined. After defining the red, green, and blue intensity profiles, type:

IDL> **tvlct, red, green, blue**

to load the color table. If you wish to read the color table which is currently loaded, type:

IDL> **tvlct, red, green, blue, /get**

2.6.7 STRETCH [IDL]

The following command can be used to invert the color table

IDL> **stretch, 255, 0**

2.6.8 GAMMA_CT [IDL]

The gamma factor for the color table can be adjusted directly using the routine GAMMA_CT. A sample call is:

IDL> **gamma_ct, 0.6**

2.6.9 MOVIE [IDL-YO]

The MOVIE routine is an IDL routine that displays an image data cube (x,y,t). The data should be byte type. It is run by typing:

IDL> **movie,data**

You can vary the speed of the movie. No information is printed with the image, for that see

STEPPER and XSTEPPER. The standard IDL routine was modified to make !ORDER default to zero which is the proper orientation for Yohkoh images.

2.6.10 PALETTE [IDL-LIB]

PALETTE is an IDL facility that allows you to create a new color table. To run it just type:

IDL> **palette**

To save the color table use TVLCT by typing

IDL> **tvlct,r,g,b,/get**

This will save the red, green, and blue vectors in *r*, *g*, and *b*. To re-load this color table, type:

IDL> **tvlct,r,g,b**

2.6.11 PROFILES [IDL]

PROFILES enables you to obtain a plot of a row or column of a two dimensional image by typing

IDL> **profiles,image**

you should have image in the format (size and normalization) that you wish first. It is a mouse driven program.

2.6.12 ZOOM [IDL-LIB]

ZOOM takes an image that is already being displayed and displays the zoomed region in a new window. The user uses the cursor and clicks with the left button to chose the center of the region to zoom (the selected region is displayed each time the left button is clicked). The middle button allows the zoom factor to be selected using a menu. The right button exits the zoom option (the zoom window is deleted upon exiting ZOOM). An example would be:

IDL> **tvscl, data(*,*,5)**
IDL> **zoom**

2.7 Routines for Image Alignment

The routines described here can generally be used on SXT or ground-based images.

2.7.1 COAL_IMAGE

COAL_IMAGE is a medium level routine to perform the scaling, alignment, and roll two images so they are co-aligned. Some sample calls are:

IDL> **data_out = coal_image(data, index_or_corn, out_corn, out_siz)**
IDL> **data_out = coal_image(data, index, out_corn, out_siz, index_out=index_out)**
IDL> **data_out = coal_image(img1, corner1, corner2, out_siz, mag=pix1/pix2)**

where the first parameter is the image which is to be changed so that it is co-aligned with another image. The second parameter can be an SXT index or the pixel address of the lower left corner relative to the sun center for the input image. The pixel address is in units of the input image pixel size. The third parameter is where you want to shift the image to, it is a two element array of the lower corner value (relative to sun center) for the output array in output pixel units. *out_siz* is the output size of the image, *mag* is the magnification factor required to convert the image to the same scale as the other image. The optional input *angle=angle* is available to correct for a roll difference between the two images.

2.7.2 FIT_LIMB

FIT_LIMB will find the center and radius of an image, even images with a partial limb. FIT_LIMB is a modified version of SXT_CENTER which will also work on non-SXT images. For non-SXT images, the pixel size must be known. The algorithm requires a VERY good initial guess to remove bad limb points. Some sample calls are:

IDL> **fit_limb, image, x, y, r, pixel_size=2.5**
IDL> **fit_limb, image, x, y, r, pixel_size=3.97, /interactive**
IDL> **fit_limb, image, x, y, r, pixel_size=2.5, initial=[434,512,403]**
IDL> **fit_limb, image, x, y, r, index=index**

where the input is *image* which is the solar image. The outputs x and y are the E/W and N/S position of the sun and r is the solar radius. The x and y locations are in full resolution pixels if it is an SXT image. It is possible to return the location in the summed pixels (half resolution or quarter resolution) by using the option */summed_pixels*. The keyword *index=index* should be used for SXT image, and the keyword *pixel_size* should be used for non-SXT images. For non-SXT images, the user should specify an initial guess using the keyword *initial* or the */interactive* option which will allow the user to click on the sun center location. There are other options and output (like the oblateness and the phase of the 2nd harmonic) which is described in more detail in the documentation header.

2.7.3 DSK_LOCB [*]

DSK_LOCB (disk locator) will find the center coordinates and disk radius of an optical solar image. It currently works on white light, Hα, and magnetogram images, or anything with a clearly defined edge. The images are all treated using a histogram technique. (There is an old routine called DSK_LOC that uses a derivative technique for white light etc., as they have a high and noisy background). A sample call is:

IDL> **dsk_locb, image, horzcnt=x0, vertcnt=y0, horzrad=rx, vertrad=ry**

where the keywords return the values of the radius and disk center.

A complete list of the optional keywords follows:

`horzcnt`	horizontal center, floating point
`vertcnt`	vertical center, floating point
`horzrad`	horizontal diameter, floating point
`vertrad`	vertical diameter, floating point
`eastlimb`	east limb, in pixels
`westlimb`	west limb, in pixels
`npole`	north pole, in pixels
`spole`	south pole, in pixels

2.7.4 FIND_LIMB

FIND_LIMB also finds the location of the center of the optical sun and it's radius. A sample calling sequence is:

IDL> **find_limb, img, x0, y0, r0**

where *img* contains the input image and *x0,y0,r0* receive the output center location and diameter of the disk. FIND_LIMB only works on whole images. Also see FIT_LIMB.

2.8 Time Routines and Conventions

There are three time conventions used in the Yohkoh database.

2.8.1 Internal Representation

The internal representation uses two variables to define the date and time of an event. One variable (integer*2) is the number of days since 1-Jan-79 (DS79) and the other (integer*4) is the number of milliseconds from the start of the day to the event (MSOD). Almost all data are saved using this internal representation which uses a total of 6 bytes.

The Yohkoh structures use the internal time representation and store the two values in tags .TIME and .DAY. The routine GT_TIME and GT_DAY will extract the data in these fields (the routines are needed because the .TIME and .DAY fields are nested in the .GEN portion of the index structures).

2.8.2 External Representation

The external representation uses a 7 element integer array to define the time of an event. This representation uses 14 bytes.

```
(0) = hour
(1) = minute
(2) = second
(3) = millisec
(4) = date
(5) = month
(6) = year
```

2.8.3 String Representation

The time of an event can also be represented with a string. The general rules are:

- The month uses the first three letters of the month (Jan, Feb, Mar...) OR it can use the month number with the / notation.
- For the standard convention using the three letters of the month:
  - The date and year are separated from the month using '-'
  - The year can come before or after the month (7-Jan-92 or 92-Jan-7)
  - If the year is not passed, it will use the current year
  - The year can be two or four digit notation
- If the month number and / notation is used:
  - It is assumes YY/MM/DD or DD/MM/YY. If you wish to use MM/DD/YY, then the /MDY switch must be used on ANYTIM2INTS, ANYTIM2EX and TIMSTR2EX.
  - There can be no spaces before or after the /
  - It must include all three items (date, month, year)
  - It differentiates between YY/MM/DD and DD/MM/YY by looking for the year, which must be greater than 1932 and less than 1999.
- The hour, minutes and seconds are separated by ':'
- The milliseconds are entered as a fraction of seconds. For example, '16-Dec-91 02:29:40.819'
- There is a space between the date and the time
- The time can come before or after the date

2.8.4 FMT_TIM

The routine FMT_TIM will take a structure and return a string with the date and time. For example, the command:

IDL> **a = fmt_tim(roadmap(0))**

will return the string '16-DEC-91 02:29:40'. It is possible to use the /MSEC keyword and get '16-DEC-91 02:29:40.819' instead.

2.8.5 FMT_TIMER

The routine FMT_TIMER is like FMT_TIM, except that FMT_TIMER is a procedure which prints the first and last time in the index structure:

IDL> **fmt_timer, index**

`(first, last) = '16-DEC-91 02:29:40', '16-DEC-91 02:29:40'` In this case, the index structure was a scalar, so the first and last times are the same. The times can be returned in optional output variables:

IDL> **fmt_timer, index, t1, t2**

The *t1* and *t2* are string variables which can be used in calls to other routines (such as PLOT_GOES, for example).

2.8.6 ANYTIM2INTS

ANYTIM2INTS takes any of the three representations and returns a structure with the internal representation. The structure has a .TIME and a .DAY field. An example is:

IDL> **a = anytim2ints('23-jun-92 3:00')**

This routine has an optional keyword input to allow the user to offset the input time by some number of seconds. If the input is an array of times, then the offset can be a scalar value or it can be an array of offsets which is the same length as the input. For example,

IDL> **a = anytim2ints(roadmap, off=60.5)**
IDL> **a = anytim2ints(roadmap, off=off)**

The following command will set up a structure which has has the dates of 100 Mondays starting on 7-Jan-91 (since 7-Jan-91 is a Monday and there are 86400 seconds in a day). The second example sets up a 500 element array with times separated by 90 seconds.

IDL> **dates = anytim2ints('7-jan-91', off=findgen(100)*86400.*7)**

IDL> **tt = anytim2ints(!stime, off=findgen(500)*90.)**

2.8.7 ANYTIM2EX

ANYTIM2EX takes any of the three representations and returns an array with the external representation. If the input is only one time, then the output is a seven element array. If it is an array of 'N' elements, then the output is a 7×N array.

IDL> **b = anytim2ex(roadmap)**

2.8.8 GT_DAY

GT_DAY is a function that returns the date from the index or roadmap structure in a string variable (other are options available) by typing:

IDL> **date = gt_day(index,/string)**

2.8.9 GT_TIME

GT_TIME is a function that returns the UT from the index or roadmap structure in a string variable (other are options available) by typing:

IDL> **time = gt_time(index,/string)**

2.8.10 INT2SECARR

INT2SECARR takes a structure (e.g., roadmap or index) containing time and will create an array of times (t) in seconds from some given reference. If the second parameter is not passed, then the first time in the input array is used

IDL> **t = int2secarr(roadmap)**

The example above will produce an array of times with respect to the first time.

IDL> **t = int2secarr(roadmap, ref_time)**
IDL> **t = int2secarr(roadmap, '1-sep-91')**

2.8.11 ANYTIM2DOY

ANYTIM2DOY takes any of the three representations and returns the day of year number.

IDL> **b = anytim2doy(roadmap)**

2.8.12 DOY2EX

DOY2EX takes a day of year and a year and returns the seven element array containing the date.

IDL> **daytim = doy2ex(doy, year)**

2.8.13 EX2DOW

EX2DOW will take the external representation and return the day of the week number, 0 for Sunday, 1 for Monday, ...

IDL> **dow = ex2dow(tarr)**

2.8.14 EX2FID

EX2FID will take the external representation date/time and return a string with the fileID.

IDL> **fid = ex2fid(input)**

2.8.15 TIM2ORBIT

TIM2ORBIT will take a set of input times and will determine which orbit and which week the data falls in. It can also return the number of minutes into the orbit for an input time. A sample interactive use of the routine would be:

IDL> **tim2orbit,'23-jun-92 6:00',/print**

The item returned for this would be:

```
     Input                S/C Day     Night in   FileID     WeekID  D/N    SAA?
   Date     Time         Time  Min Ago  (min)
23-JUN-92  06:00:00   05:13:14 46.77   14.98  920623.0513  92_26  Day
```

The S/C day started 46.77 minutes earlier, and night will occur in 14.98 minutes (it is S/C day mode). The FileID for this time is 920623.0513 and the WeekID is 92_26.

Optional Keyword Output Parameters (the number of output elements is the same as the input):

`FEM`	the FEM structure
`FID`	the FileID for each input time
`WID`	the WeekID for each input time
`SAA`	a boolean array, TRUE if in SAA
`SCDAY`	a boolean array, TRUE if in S/C Day
`TIM2FMS`	the number of minutes since sunrise
`TIM2NIGHT`	the number of minutes before sunset

2.8.16 SEL_TIMRANGE

Given an array of times, SEL_TIMRANGE will return the subscripts of the elements which fall between a set of input times. The */between* option should be used when you want all datasets between the start and stop times, since the default is to return the first entry BEFORE the start time (this was for historical reasons when accessing the FEM structures). It is possible to have a boolean array returned which is the same length as the input. The array will have a 1 if the input time is in the time range specified, and a 0 otherwise. This is useful for using with WHERE commands.

IDL> **ss = sel_timrange(roadmap, st_tim, en_tim)**
IDL> **ss = sel_timrange(orbit_arr, '1-jul-92', '22-jul-92', /between)**
IDL> **ok = sel_timrange(index, '23-jun-93', '24-jun-93', /boolean)**

2.8.17 PR_TIM2WEEK

PR_TIM2WEEK will convert a Yohkoh time into the corresponding week ID and print a message on the screen. For example:

IDL> **pr_tim2week** ; Give the current week number
IDL> **pr_tim2week,'15-nov-91'**
IDL> **pr_tim2week,'15-nov-91',/range**

Setting the */range* keyword will cause the routine to display the first and last day of the week. Short weeks (the first or last week of the year) are flagged with an asterisk. PR_TIM2WEEK calls ANYTIM2WEEKID to make the conversion.

2.8.18 PR_WEEK2TIM

PR_WEEK2TIM will print on the screen the first and last day of a specified week number (integer) or Yohkoh week ID (string). For example:

IDL> **pr_week2tim** ; List current Yohkoh week

```
IDL> pr_week2tim,'91_46'
IDL> pr_week2tim,'91_46a'          ; Last letter is optional
IDL> pr_week2tim, 4                ; Week 4 of current year
IDL> pr_week2tim, 4, year=93       ; Week 4 of 1993
```

Short weeks (the first or last week of the year) are flagged with an asterisk. PR_WEEK2TIM calls WEEKID2EX to make the conversion.

2.9 Routines for Windows and Output Devices

2.9.1 Making Landscape and Portrait Plots

The command to have the PostScript plots be in landscape orientation is:

```
IDL> set_plot,'ps'
IDL> device, /land
```

In order to have the portrait plots take up the full page, there is an un-official routine called LASER. The calling sequence is:

```
IDL> set_plot,'ps'
IDL> laser, /portrait
```

2.9.2 WDEF

WDEF creates a window of a set size. The first example will make a 512×512 window in the top right hand corner of your screen. The second example makes a 128×128 window, and the last example makes a 900×512.

```
IDL> wdef,0
IDL> wdef,0,128
IDL> wdef,0,900,512
```

Window 1 is in the lower right, 2 is in the upper left, and 3 is in the lower left (even windows on top, window modulo 4 equaling 0 or 1 is on the right)

2.9.3 WSHOW [IDL]

WSHOW pushes a covered IDL image window to the foreground while keeping the text window active (this is not necessarily true for cases where the active window is selected by the location of the cursor). Just type

```
IDL> wshow
```

2.9.4 HARDCOPY

Dumps the contents of a 'X' screen to a laser printer with either b/w or color options by simply typing:

IDL> **hardcopy**
IDL> **hardcopy, /black**

It is recommended to only use HARDCOPY to get hardcopies for images. If you want to get a hardcopy of a line plot, then first change the output device to a PostScript file (idl.ps), re-run the plotting program or plotting commands, and then issue the print command. An example would be:

IDL> **set_plot, 'ps'**
IDL> **.run plot_sot**
IDL> **pprint**
IDL> **set_plot, 'x'**

2.9.5 PPRINT

PPRINT closes and prints an idl.ps file. An example of how to use it would be

IDL> **set_plot,'ps'** ; opens idl.ps file
IDL> **device, /land** ; optionally use landscape orientation
IDL> **.... (your plot commands)**
IDL> **pprint**
IDL> **set_plot,'x'**

This will send the plot to the laser printer.

2.9.6 SETDISP

It is possible to issue the "setenv DISPLAY" command from within IDL by using the routine SETDISP (if the user forgets prior to starting IDL). A sample call would be:

IDL> **setdisp, '192.68.162.109:0'**

3 Low Level Routines

3.1 Plotting Routines

3.1.1 CLEARPLOT

CLEARPLOT will reset the IDL plotting variables.

3.1.2 CLEAR_UTPLOT

CLEAR_UTPLOT will reset the UTPLOT plotting variables.

3.2 File and Directory Routines

3.2.1 CONCAT_DIR

CONCAT_DIR is fundamental in allowing software to be portable between Unix and VMS systems. Given a directory logical name (or environment variable) it will figure out whether to put a '/' between the directory and the file name (for Unix systems), or to use a ':' (for VMS systems). A restriction is that the directory must be a logical or environment variable. Some sample calls are:

IDL> **outfil = concat_dir(dir, filnam)**
IDL> **outfil = concat_dir('$DIR_GEN_OBS', 'obs92_23a.01')**

3.2.2 BREAK_PATH

Not sure why this was missing!!!!!

3.2.3 FILE_LIST

FILE_LIST enables you to get a string array *infil* with all the files that fulfill a set of specifications by typing:

IDL> **infil = file_list(dir, filenam)**

IDL> **infil = file_list(dir, filenam, /cd)**

where *dir* is a string array of directory name(s) and *filenam* is a string with the file designation (wild cards are ok — e.g., sfd* or bda920101.*).

3.2.4 FILE_MENU

FILE_MENU works very much like FILE_LIST but is menu driven. It only gets you a single file (infil) by typing:

IDL> **infil = file_menu(dir,filenam)**

3.2.5 DATA_PATHS

DATA_PATHS is fundamental to several routines including YODAT. The routine should return a string array with all the directories where there are Yohkoh reformatted data. The routine is site specific since each institute organizes the data slightly differently. A common practice is to use DATA_PATHS2 to get the standard data directories, and then append the specialized data directories. See the sample DATA_PATHS program in the 'site' branch of the Yohkoh software.

3.2.6 DATA_PATHS2

DATA_PATHS2 searches all '/yd' directories for directories with YY_WW names (for example, '/yd1/92_36a' or '/yd10/91_35a'). Since the generation of the data directory list is done automatically, it is not necessary for a programmer to edit DATA_PATHS every time a new week of data are placed on-line.

3.2.7 NEW_DPATH

NEW_DPATH will add elements to the Yohkoh data paths array. These are the disk areas automatically searched for reformatted files. A file option allows reading from a file of pathnames. This can be conveniently called from an a personal IDL startup file to automatically define paths when entering IDL. Switches are available to add current directory, initialize contents, remove elements, or save current contents for later input via file option.

IDL> **new_dpath, patharray**
IDL> **new_dpath, /current**
IDL> **new_dpath, /init**
IDL> **new_dpath, /file**
IDL> **new_dpath, file=file**
IDL> **new_dpath, /save**

Current contents of data paths area may be reviewed with the command:

IDL> **prstr, data_paths()**

3.2.8 GBO_PATHS

GBO_PATHS is similar to DATA_PATHS, but it looks for the environment variables beginning with 'DIR_GBO' and creates a list. This routine is used by YODAT to find where the ground-based FITS files are located.

3.2.9 GET_SUBDIRS

GET_SUBDIRS will return all the subdirectories under a given directory. An example:

IDL> **dirs = get_subdirs('/ys')**

3.2.10 PR_PATH

PR_PATH will display all the directories that are in the IDL variable !path.

3.2.11 PATH_LIB

PATH_LIB will search directories in the IDL !path variable in the order of the list to find a given IDL program. The default is to only give the first instance that the file is found, but it is possible to get all locations by using the */multi* switch. A wild card ('*') effectively sets the */multi* switch). Same examples are:

IDL> **path = path_lib()**
IDL> **path = path_lib('add*')**
IDL> **path = path_lib('yodat', /multi)**

3.2.12 PR_PATH_LIB

PR_PATH_LIB is like PATH_LIB, except PR_PATH_LIB is a procedure which will print the results to your screen. For example,

IDL> **pr_path_lib,'gt_*'**

will return a list of all "gt_" routines

3.2.13 FILES_SEARCH

FILES_SEARCH will search all files in sub-directories from a given directory for a given string. In the following example, all .PRO files under the /ys tree are searched to find the string 'CONV_'.

IDL> **files_search, '/ys', 'CONV_'**

3.3 String Manipulation

3.3.1 PRSTR

PRSTR prints any string array to the terminal or a file, one element per line. For terminal output, the default is to print by screen (Unix "more" behavior). The */compress* switch removes null lines and excess white space prior to output. This is useful (for example) for printing FITS headers to terminal. */hc* will create a hard copy to the printer selected by PPRINT. *file=file* will write the results to a file.

IDL> **prstr, strarray**
IDL> **prstr, strarray, /print**
IDL> **prstr, strarray, /compress**
IDL> **prstr, strarray, lun**

3.3.2 STR2ARR

Given a string, STR2ARR will separate the items into an array using a ',' as the delimiter by default. Some examples are:

IDL> **arr = str2arr('a,b,c,d,e,f,ggg,hh')**
IDL> **arr = str2arr(!path, delim=':')**

3.3.3 ARR2STR

Given an array, ARR2STR converts it to a string. The items will be separated by a comma by default.

IDL> **str = arr2str(indgen(10))**

3.3.4 STR_REPLACE

Given a string (or an array of strings) it is possible to replace all occurrences of a given string using STR_REPLACE. In the example below, all occurrences of "sfr" are replaced with "spr".

IDL> **out = str_replace(in, 'sfr', 'spr')**

3.3.5 WC_WHERE

The WC_WHERE function will search a string array for the occurrence of a substring and return the indices of the elements where a match is found. The substring may contain one or more wild cards. An optional third parameter contains a count of the elements returned. The *case_ignore* switch may be used to force the search to be insensitive to the case of the characters.

IDL> **ss=wc_where(strarray, pattern)**
IDL> **ss=wc_where(filearray, 'sfr9310*', filecount)**
IDL> **ss=wc_where(strarray, '*time*',/case_ignore)**

3.4 Miscellaneous

3.4.1 REBIN [IDL]

REBIN is an IDL routine that enables you to make a 2 dimension array larger or smaller (it also works on multidimensional arrays). To use it to create a new 512×512 image (ndat) from a 64×64 image (data) type:

IDL> **ndat=rebin(data,512,512,/sample)**

Note that the new x and y dimensions must be integral multiples (or divisors) of the original array dimensions.

3.4.2 PLOT_TRAV

It is possible to plot a time line showing when the SXT and BCS team members will be on travel to Japan by using PLOT_TRAV. The plot separates the people by institution. Some examples are:

IDL> **plot_trav**
IDL> **plot_trav, /lparl**
IDL> **plot_trav, /ops**
IDL> **plot_trav, /bcs**

The default is to plot SXT personnel. To show BCS personnel use the */bcs* switch. The */lparl* switch will only plot the key Lockheed SXT personnel.

3.4.3 RD_TFILE (How to Read Text Files with IDL)

RD_TFILE is a generic ASCII (text) file reading routine. It can be used to return the contents of any text file as an IDL array. A fast algorithm is used which makes it more than an order of magnitude faster than reading line by line until the end of the file. It can optionally read and convert string or numeric tables. It is ideally suited for use in reading calibration files – in many cases, existing ASCII calibration files can be read without change. The automatic comment elimination feature makes it simple to internally document calibration files or add to existing documentation without any changes to the IDL access routines.

It is hoped that the use of specialized ASCII read routines can be minimized by use of this routine. Even if a calibration file has a complex internal structure, RD_TFILE can perform the initial read and the user can handle special formatting problems with the IDL intrinsic READS function (read from the string array returned by RD_TFILE). No explicit file open or close commands would be required by those routines which use RD_TFILE. Other typical uses are reading IDL source code (extract documentation), reading in Unix script and VMS command files for parsing, etc.

The following features are available:

- read ASCII files into IDL variables
- read table data using default or user specified delimiter
- convert ASCII numeric vector or table data
- remove partial and full comment lines using default or user specified comment delimiter
- optionally remove excess white space and null strings

Calling Examples

IDL> **text=rd_tfile(filename)** ; orig. file-> string array
IDL> **text=rd_tfile(filename,/nocomment)** ; same less comment lines
IDL> **text=rd_tfile(filename,/compress)** ; same less excess blanks
IDL> **data=rd_tfile(filename,/hskip,head=head)** ; return file header in head
IDL> **data=rd_tfile('text.dat',3)** ; strarr(3,N) (table data)
IDL> **data=rd_tfile('fdata.dat',/auto,/convert)** ; auto determine #col/data-type

As a specific example, you can read in a Unix password file as a table with a single command:

IDL> **passwd=rd_tfile('/etc/passwd',delim=':',/auto,/nocomment)**
IDL> **help,passwd**

```
PASSWD          STRING    = Array(7, 39)
```

Now, search for given user using WC_WHERE and print out relevant fields

IDL> **ss=wc_where(passwd(0,*),'free*')** ; search on name field
IDL> **prstr,passwd(*,ss)** ; print users record

```
freeland
YI/g8JD8wq0uo
5498
```

```
15
Samuel L. Freeland
/sxt1data0/people/freeland
/bin/tcsh
```

3.4.4 DEF_TAPD

DEF_TAPD returns default tape drive names as a scaler string. It includes default entries for the most common operating systems. It offers automatic substitution of the an input drive number (default is drive 0). If the system variable !ys_deftape is defined, the routine will return that value (with drive number substitution, if requested).

IDL> **drive = def_tapd()**
IDL> **drive = def_tapd(2)**

3.4.5 WMENU_SEL

WMENU_SEL is an expansion on the IDL WMENU routine. It allows for many menu pages if the number of items will not fit on one page. It also allows a user to select several items. Some sample calls are:

IDL> **ss = wmenu_sel(array)**
IDL> **ss = wmenu_sel(files, /one)**

A Installation and Maintenance of the Yohkoh Software

A.1 Network Installation

This section describes the network installation of compressed tar files on Unix systems[1]. Current disk allocation requirements: Approximately 100 MBytes which includes the Software and several supporting databases (35 Mbytes for only IDL and FORTRAN software or 25 Mbytes for only IDL software).

1. Determine which user will own the Yohkoh software and database area. Some sites prefer to create a special "Yohkoh" account for management but this is not an installation requirement. Log into the appropriate account.
2. Determine the path where you want to locate the Yohkoh Software. Let's call this path *yparent*. This pathname should be an nfs name per /etc/fstab if the software is to be shared among machines. Of course, the owner of the Yohkoh Software (determined in preceding step) requires write access to *yparent*.
3. Transfer the boot-up script via anonymous FTP.

% **mkdir -p** *yparent***/ysw/swmaint/tar**	# local directory for tar files
% **cd** *yparent***/ysw/swmaint/tar**	# move there
% **ftp 133.74.8.100**	# user=ftp, passwd=name@node
FTP> **cd pub/swmaint/tar**	# remote location (isass0)
FTP> **get ys_install**	# boot up script
FTP> **quit**	

4. Install the software and standard database using the script *ys_install*. The defaults are usually ok, but see script header for available options. Use the */HELP* switch to get updated instructions and the latest Yohkoh Software and Database descriptions without installing anything. (current directory is still = *yparent*/ysw/swmaint/tar).

% **csh ys_install**	# INSTALL, take defaults
% **csh ys_install [OPTIONS]**	# INSTALL, use 1 or more options
% **csh ys_install /HELP**	# No installation, Help Information

Section B.1 describes the procedure for setting up the Yohkoh environment for new accounts.

A.2 Network Upgrades

This section is applicable if you have previously installed the Yohkoh Software on your Unix workstation. It will upgrade the latest version of the software and database files using the existing locations.

[1] If you have already installed Yohkoh Software in the past and do not need to relocate it, skip to section A.2. (Upgrades) instructions.

NOTE: beginning with Yohkoh Software Version# 931107.02, auto-upgrade scripts are included in the distribution. Once you have upgraded to this level, all FTP commands are automatically issued. For older Versions (before 930513.01) you must first retrieve the installation "boot" script – other files will be auto-FTPed as required.

% **tail -1 $ys/gen/setup/ys.version** # determine current version number

A.2.1 New Versions (starting with 931107.02)

Log in as owner of the Yohkoh Software tree; verify that $ys is defined. For possible options see header of $ys/gen/script/ys_install. Next, [2]

% **csh $ys/gen/script/ys_install**
% **csh $ys/gen/script/ys_install [OPTIONS]**

A.2.2 Old Versions (Before 930513.01)

Log in as owner of the Yohkoh Software tree; verify that $ys is defined

% **cd $ys/swmaint/tar** # local tar site
% **ftp 133.74.8.100** # user=ftp, passwd=name@node
FTP> **cd pub/swmaint/tar** # remote location (isass0)
FTP> **get ys_install** # boot up script
FTP> **quit**

see header of $ys/swmaint/tar/ys_install for options, then:

% **csh $ys/swmaint/tar/ys_install**
% **csh $ys/swmaint/tar/ys_install [OPTIONS]**

A.3 Automatic Upgrades of Yohkoh Software

The command to automatically upgrade the Yohkoh software on a periodic basis may be inserted into a "cron job" at remote sites[3]. If you are not familiar with function and operation of cron jobs on your particular machine, you should consult with your system manager. Although there may be some system dependencies not covered here we present some sample implementations here. Notes: *ys* is the path assigned to $ys ; *ysowner* is the username of the Yohkoh Software owner.

[2]this command may be placed in a daily "cronjob" to provide automatic upgrades – see discussion on page 34.
[3]Requires Yohkoh Software Version: 931108.02 or higher

SUN/Mips/SGI/OSF

1. Log on as the owner of the Yohkoh Software
2. Edit the "crontab" file with the command:

   % **crontab -e**

   and add a line with the format:
   `MM HH * * * setenv ys` *ys*`;csh $ys/gen/script/ys_install`
3. A specific example which upgrades the software daily at 23:00 local time is:
   `00 23 * * * setenv ys /user1/ysw; csh $ys/gen/script/ys_install`

Ultrix

1. Log on root
2. Add the following line to /etc/crontab via editor
   `MM HH * * * rsh -l` *ysowner* `"setenv ys` *ys*`;csh $ys/gen/script/ys_install"`
3. A specific example which upgrades the software daily at 2:00 local time is:
   `00 02 * * * rsh -l ysoft "setenv ys /user1/ysw; csh $ys/gen/script/ys_install"`

For Ultrix systems that wish to use the root account to run the batch job, you can exclude the "rsh -l *ysowner*" portion of the command. If you do not run as the root user, then you will also need to edit the `~/.rhost` file for the account which will be used to execute the "ys_install" script, and add a line similar to the following line, except with your node name.

```
isass0.solar.isas.ac.jp         root
```

A.4 Site Specific Cleanup Script

Sites may choose to execute a customized cleanup script following a Yohkoh installation. Common uses for this could be to set file protections, ownerships, group access etc. (chmod, chown, chgrp). It is possible to have such a script run automatically as part of the installation/upgrade procedure by naming the script $ys/site/script/ys_install.local. If this file exists, the Yohkoh Installation scripts will source it following the successful completion of an upgrade and/or installation. It is up to the local manager to verify that all commands within this script have the correct syntax. For example, if *chown* commands are issued, then the Yohkoh installation procedure would have to be run from the root account to have these take effect.

A.5 DATA_PATHS

Remote sites have many differences in available disk space, physical disk names and path names. To allow seemless software access to Yohkoh data and database files, we make use of Unix environmentals and VMS logicals within the routines. This allows relocation of existing data areas and addition of new data areas with no changes required to the access software.

Reformatted data access routines use logicals or environmentals which are named "ydNN" where "yd" stands for Yohkoh data and NN is an integer. Every site should define at least one data area (usually yd0); there is no upper limit on the number. The most common scheme is to have one area (and one environmental) per physical disk available for Yohkoh reformatted data.

To add a new area (ie, add a new environmental), a line should be added to the file $ys/site/setup/setup_paths. This script file is one of a few which are automatically executed during the Yohkoh IDL setup to configure the local and Yohkoh environment.

```
Each line is a Unix 'setenv' command of the form:

           setenv  yd<N>  <pathname>
WHERE:
<N>        = integer = disk area number (traditional to start at zero (0) )
<pathname> = pathname of data area to add
             If the area is to be shared via NFS, this should include the
             mount name as found in /etc/fstab
```

For example, appending the following lines to $ys/site/setup/setup_paths would add 3 reformatted data areas.

```
#....
#..... current contents of file: $ys/site/setup/setup_paths
#.....
# add 3 data areas for Yohkoh reformatted data
setenv yd0     /user1/yohkoh
setenv yd1     /data_disk2/yohkoh
setenv yd2     /machine_mount2/scratch/yohkoh
#..............................................
```

These areas will then be searched by the Yohkoh Software for reformatted data files. (creating the directories and placing data there is not covered here).

A.6 Receiving and Installing Secondary Yohkoh Database Files

A summary of supporting and secondary databases and the approximate size is provided in the following table. Certain database sets are automatically included when a Software installation or full upgrade is performed (described in previous section). All database files are also available on Exabyte tape ("ydb tape") To receive a tape containing the mission-long Yohkoh database and to be placed on the distribution list for future releases, send e-mail:

```
  TO: software@isass0.solar.isas.ac.jp
SUBJ: ydb tape request
```

The IDL routine RD_YDBTAP (described in the User's Guide) can be used to extract and install user specified portions (or all) of the Yohkoh database from the ydb tape. That routine will optionally provide assistance in setting up and customizing the local environment prior to extraction and installation. The physical disk locations where each database resides at any given site are flexible since all software references are through the use of environmentals (Unix) or logicals (VMS). The IDL routine HELP_PREFIX provides some additional information about the specific database contents.

This table can be used to estimate current and future disk space requirements for the Yohkoh database. Unless otherwise noted, the distributed files contain one week of data. Column one (PFX) indicates the three letter prefix associated with each database set. Items marked with an asterisk are included in the automatic software installation procedure described in the preceding sections.

PFX	Contents	Avg File Size		Mission Size (Thru 1/94)
ATR	Raw S/C Attitude Database	2684 Kb		314.6 Mb
ATT	Processed S/C Attitude Database	2024 Kb		237.2 Mb
EVN*	Yohkoh Event Log	14 Kb		1.8 Mb
FEM*	Yohkoh Orbital Ephemeris	7 Kb		1.2 Mb
G6D	GOES 3 Sec Light Curve for S/C 6	2725 Kb		181.0 Mb
G7D	GOES 3 Sec Light Curve for S/C 7	2742 Kb		182.2 Mb
GBE	GRO BATSE Event Log	19525 Kb	[1]	19.1 Mb
GBL*	GRO BATSE Light Curves	143 Kb		19.1 Mb
GEV*	GOES Event Log	5 Kb		0.7 Mb
GXD	GOES Derived One Minute Light Curve Data	5 Kb		0.0 Mb
GXT*	GOES One Minute Light Curve Data	330 Kb		40.0 Mb
NAR*	NOAA Active Region	3 Kb		0.4 Mb
OBD	BCS Observing Log	690 Kb		81.7 Mb
OSF	SXT Full Frame Observing Log	52 Kb		6.3 Mb
OSP	SXT Partial Frame Observing Log	385 Kb		45.6 Mb
OWH	WBS/HXT Observing Log	1266 Kb		150.9 Mb
SDC	SXT Dark Current Images	1827 Kb		201.7 Mb
SDW	SXT Dark Current Images for Warm CCD	5196 Kb		81.2 Mb
SFC	SXT FFI Calibration (Leak) Image	528 Kb		4.2 Mb
SFM	SXT FFI Monthly File (daily SFD image)	6506 Kb	[M]	158.9 Mb
SSC	SXT Synoptic Images (centered)	697 Kb		77.6 Mb
SSE	SXT Synoptic Images (east of center)	703 Kb		(inc)
SSW	SXT Synoptic Images (west of center)	703 Kb		(inc)
XAD*	Exabyte ASCII Directory for Archive Tape	10 Kb		2.1 Mb
XBD*	Exabyte Binary Directory for Archive Tape	7 Kb		1.5 Mb

[1] - only 1 file for mission
[M] - each file contains one month of data

The following table indicates which datasets are important for various purposes.

Description	Datasets
Critical general datasets	FEM
Critical datasets for SXT calibration	SDC, SFC
Nice SXT datasets to have	SSC/E/W, SFM
Critical datasets for alignment	ATT
Non-necessary datasets for alignment	PNT, ATR
Observing log and event logs – nice to have	OBD, OSP, OSF, OHW, EVN
Sets that make reading Exabytes easier	XBD, XAD
Nice sets to have (don't take much space)	GEV, GXT, GBE, GBL
Nice sets to have (take much space)	NAR, G6D, G7D

A.7 Customizing the Overall Yohkoh Environment

The Yohkoh Software system allows customization of both site and user level IDL and operating system environments . These features are optional, and many users may work effectively without any knowledge or implementation of these features. On the other hand, as the Yohkoh system evolves at a given site, some of these features may be required to allow the Yohkoh system manager to tailor the system for the given site resources. For example, the multitude of Yohkoh ancillary databases may be divided among multiple disks or machines to match local disk resources. Sophisticated users may use some of these features to further customize the IDL environment while still remaining within the Yohkoh framework.

A.7.1 Use of Environmentals/Logicals for Yohkoh Software and Database

All supported Yohkoh Software uses Unix environmentals and VMS logicals for data and database files. The Yohkoh Software tree is also defined in terms of environmental variables. This allows relocation of Yohkoh Software and/or database files to different paths, disks, or machines without any changes required in the source code and may occur transparently to the Yohkoh Software user. It also minimizes restrictions on disk space allocation as different sites may organize the Yohkoh databases to suit the local resources.

A.7.2 Yohkoh environmental configuration files

The Yohkoh OS and IDL environment is established through the execution of several Unix script (or VMS command) files and IDL startup files. A single script acts to control the flow of script and setup execution. In some cases, multiple versions of scripts may exist in a general, site, and personal areas to allow customization according to the following execution order.

	Type	Path	Function
1)	GEN	$ys/gen/setup	setup default Yohkoh environment
2)	SITE	$ys/site/setup	allow site modifications to the above
3)	PERS	$HOME	allow personal modifications to the above

The files described below are used to configure the Yohkoh environment. Files for which a SITE version is maintained may be used by the site Yohkoh Software manager to customize the site environment. If no site version currently exists, the corresponding file from gen ($ys/gen/setup) may be used as an example or template (ie, copy to $ys/site/setup and edit). Note that files in SITE are not overwritten by a Yohkoh Software upgrade so site changes are not clobbered (unless requested).

File	Function	GEN	SITE	PERS
idl_setup	control flow of setup execution	yes	no	no
setup_paths	set top level database locations	yes(1)	yes	no
setup_dirs	define specific database paths	yes	yes	no
setup_alias	define OS symbols / aliases	yes	yes	no
setup_mail	define mailing list env	yes	yes	no
setup_ysenv	other env/alias customization	yes	yes	yes
IDL_STARTUP	define Yohkoh IDL environment	yes	yes(2)	yes(3)

1. the version in gen is not executed – it is provided as a template for site managers (some internal documentation)
2. The file $ys/site/setup/IDL_STARTUP (maintained by the site manager) is executed if it exists.
3. If user has the $IDL_STARTUP environmental defined, it will be executed following the Yohkoh-wide IDL_STARTUP. See the IDL manual for a description of IDL_STARTUP.

A.7.3 setup_paths – Yohkoh Database and Reformatted Data Locations

The site version of setup_paths may be used to split the Yohkoh database across multiple disks or machines. It is also used to define the areas where the software will search for reformatted files. It resolves top level environmentals with site specific path names. Local pathnames should include NFS names (per fstab) if database or reformatted files are shared among machines. The file: $ys/gen/setup/setup_paths may be used as a template if $ys/site/setup/setup_paths does not already exist. It includes some internal documentation and examples.

A.7.4 setup_dirs

This script defines specific database environmentals in terms of the top level environmentals defined in setup_paths. The site version of this file is only required if the Yohkoh database is split. As with setup_paths, this is just a series of Unix setenv commands (or VMS DEFINE commands). We recommend not including path references in this file – instead, the definitions should be in terms of the environmentals (or logicals) defined in setup_paths.

A.7.5 setup_alias

This script provides a convenient method to define aliases or symbols which can be used by all site users. This can provide a common OS environment at the site.

A.7.6 setup_ysenv

The intended use of this script is to set up several Yohkoh environmentals which control IDL startup. These include setting defaults for X windows startup, message of the day display, fast startup, etc. Use the "gen" version of the file ($ys/gen/setup/setup_ysenv) as a template and for some discussion of options. The site manager may use the site version of this file to change the distributed defaults. If a user has a file with this name in the HOME directory, it will be executed to supercede or supplement the default and site environment. Since it is just a script file which gets automatically executed during the Yohkoh startup, sophisticated users may discover additional uses.

A.7.7 Multiple Software Host and Software Relocation

Large, multi-user Yohkoh sites may consider maintaining multiple sets of the Yohkoh Software on different machines. The main purpose of such a system is to allow software access from a backup system(s) in the event that the primary system is off-line (e.g., for maintenance, repairs, etc). The Yohkoh Software distribution includes scripts which allow automatic selection of a host. The file: $ys/site/setup/yspaths.config is a data file which is used by the Yohkoh startup script to dynamically assign the location of the Yohkoh Software system. This file is created during the installation procedure and initially just includes an entry for the machine and path where the Yohkoh Software is located. The script *ys_newuser* (see Section B.1) copies this to the home directory of Yohkoh users to allow access to the Yohkoh setup scripts. To define additional hosts at a given site, the Yohkoh site manager can edit the file: $ys/site/setup/yspaths.config. One line containing two fields should be added. The file is of the form:

```
host1    path1          # original contents
host2    path2          # second host added
(host3)  (path3)
```

Where *hostN* is the hostname and *pathN* is the path location of Yohkoh Software. An example of this file is

```
isass0  /0p/yohkoh
isass2  /2p/yohkoh_sw
```

A.8 Initial Installation Using Exabyte Tapes

The large majority of users are currently performing initial Yohkoh Software installation via the network install. If a network installation is not practical, send e-mail:

```
TO:   software@isass0.solar.isas.ac.jp
SUBJ: tape installation procedure
```

We offer an Exabyte "boot up" tape and installation instructions.

A.9 VMS Installations

The Yohkoh Software System is designed to be compatible with the VMS operating system and is currently running under VMS at several locations, however, we recommend that Unix be selected for analysis of Yohkoh data if possible.

Currently, the VMS installation and upgrade process is not as automated as the Unix procedure. At the time of publication, the BCS team was the main user of the Yohkoh package under VMS, and it appears that they will transition completely to Unix in the future.

The BCS team may be able to offer some guidance and VMS command files to assist in installing the Yohkoh system under VMS, however we can make no promises that future Yohkoh software releases will be supported on VMS systems. For more information, send e-mail and it will be forwarded appropriately.

```
TO:   software@isass0.solar.isas.ac.jp
SUBJ: VMS installation procedure
```

A.10 Explanation of Software Release Version Numbers

Yohkoh Software version numbers are used for system maintenance and to facilitate problem reporting and tracking. The version numbers consist of two fields of the form:

```
YYMMDD.NN         Where:  YYMMDD = Date of Release
                              NN = Major Release Number
```

There are two types of "Releases", both of which are accompanied by an update to the date field. A "Major Release", accompanied by an increment of "NN", is generated when there is a major restructuring (deletions, renaming, moves) of the Yohkoh tree. A "Major Release" requires that remote sites execute a full upgrade (which copies over all of the *ys* tar files except *site*). A "Minor Release" is generated when there are a large number of additions to the tree, but not other restructuring. Remote sites can upgrade to new minor release levels with an incremental file – a minor release is a method of maintaining the incremental files at reasonable sizes. Upgrades to new major and minor release levels are both automatically accomplished by the upgrade scripts *ys_install*. Version numbers at remote sites are compared to the version numbers on the master copy (currently isass0) and the appropriate files are transferred and installed. To determine the Yohkoh Software version you are running type:

% **tail -1 $ys/gen/setup/ys.version**

When reporting problems, please include the Software version number within the message.

B Customizing Your Personal Yohkoh Working Environment

This section briefly describes options available for customizing the Yohkoh data analysis environment. To gain a deeper understanding of the IDL related items listed here, the user is referred to to the IDL User's Guide, Chapter: RUNNING IDL.

B.1 Initial User Setup (.yslogin)

For new user accounts, execute the following procedure one time.[4] First, determine the location of the Yohkoh software at your site. Call this path *ysw*.

% **setenv ys** *ysw*
% **source $ys/gen/script/ys_newuser**

Once this procedure has been executed, the following command will define the Yohkoh IDL environment:

% **source ~/.yslogin**

Most users will find it convenient to have this Yohkoh setup done automatically as part of the login procedure. For Ultrix and IRIX users, insert the line into your ~/.login file. Users of SUN, Mips, and most others should insert the line into the ~/.cshrc file.

B.2 Running IDL (run_idl)

The file $ys/gen/setup/run_idl is a script which performs the Yohkoh IDL startup procedure. When working within the Yohkoh environment, this script is run when a user enters the command `idl`. The alias *idl* is defined during the execution of idl_setup.

The following functions are performed by the run_idl script:

- define the IDL SW !path (Concatenation of Yohkoh, Personal, and IDL paths)
- setup remote X display if not already set (selected systems)
- display message of the day
- execute the Yohkoh wide IDL_STARTUP
- optionally execute user defined IDL_STARTUP
- enter IDL

The following additional options are available by entering a second parameter on the command line:

[4] Since the Yohkoh Software at a site is often shared among many machines, we supply an initialization script to dynamically define the location of the Yohkoh Software. This will avoid workstation hangups when the Yohkoh server is down and also permits multiple Yohkoh hosts (page 40).

```
% idl nox       # bypass X windows setup - useful to avoid color table
                # corruption caused by 2nd IDL session or to speed up remote

% idl tek       # startup with plot device set to 'TEK' (Tektronics mode)

% idl noys      # do not include Yohkoh paths in IDL !path setup
                # for non-Yohkoh IDL work(?!)

% idl fast      # define path from file (faster) [if supported at site]

% idl ahead     # place personal path ($IDL_PATH)  ahead of Yohkoh paths
                # in IDL !path variable - default is Yohkoh paths first
```

B.3 Personal SETUP_YSENV

Under Unix, the Yohkoh startup will automatically execute (source) a script file in the users home directory which is named setup_ysenv. This can be used to personalize the operating system environment. One simple use of this file would be to alter some of the Yohkoh environmental startup defaults; see $ys/gen/setup/setup_ysenv for a convenient template and documentation.

B.4 IDL_PATH

Users may define one or more personal paths which may be used in conjunction with the extensive Yohkoh path definition. This will permit use of local and personal IDL libraries while still remaining within the Yohkoh analysis framework. To use this feature, the user should define the Unix environmental or VMS logical: IDL_PATH. See the IDL User's Guide, Chapter "RUNNING IDL" for a discussion of the Unix and VMS IDL_PATH syntax. An example of a command located in the `.login` file which will add all directories which have .PRO files under /2p/morrison/soft and /usr/local/lib/idl/lib is:

```
setenv IDL_PATH +/2p/morrison/soft:+/usr/local/lib/idl/lib
```

B.5 Personal IDL_STARTUP

Yohkoh users may define an IDL startup routine by defining the Unix environmental or VMS logical IDL_STARTUP. An IDL startup is a file which contains IDL commands. These commands are automatically executed at startup time to allow customization of the IDL environment. If IDL_STARTUP is defined, it is executed after the Yohkoh-wide and site IDL_STARTUP files so the user values will supercede the defaults. An example of a command located in the `.login` file which will cause the IDL commands in `/2p/morrison/idl_startup.pro` to be executed when IDL starts is:

```
setenv IDL_STARTUP /2p/morrison/idl_startup.pro
```

B.6 Multiple SW Host capability

See the description on page 40 for details on how to set up your system so that it can use a backup machine in the case when the primary machine is unavailable.

B.7 Switch between primary and backup Yohkoh Hosts

For sites which support multiple software hosts, the following Unix commands are available. Switch to the backup host can be done for backup system testing or in the event that the primary host has 'gone down' after your session has already started (automatic host selection is done when your session starts)

```
% idlback                # select backup host
% idlnorm                # select primary host (startup value)
```

B.8 setenv DISPLAY

It is possible to log onto a remote machine to run IDL and to have the X-window display redirected back to your local machine. See the description in the Appendix of the User's Guide for details for the necessary commands.

C Using the Yohkoh Package from a Remote Node

C.1 Logging in from a remote node

There are several ways to log into a remote node. Here are a few of them. From Unix or DEC-Ultrix:

% **telnet node_name**
% **telnet isass0**
% **telnet node_number**
% **telnet 192.68.162.109**
% **rlogin node**
% **rlogin isass2**
% **rlogin node -l username**
% **rlogin isass2 -l morrison**

From DEC-Ultrix:

% **dlogin node**
% **dlogin 24707::**
% **dlogin 24.131**

From VMS:

$ **set host node**
$ **set host 24707**
$ **set host sag**

When using TELNET, DLOGIN, and SET HOST you will be prompted for the username and password of the account you want to log into. For RLOGIN, it will check the file .rhosts to see if the account and machine that you are logging in from should automatically log you into the remote account. A sample .rhost line is:

```
sxt1.space.lockheed.com         morrison        #  DECstation 5000
```

C.2 Redirecting X-Window Outputs

If you have logged into a remote computer from a workstation, it is possible to redefine the machine that will display X-window output. The default for IDL is to make X-windows for plots and image displays. Two things have to be done before you can redirect the X-windows output: first the output device needs to be defined to be the local node, and second the security for the local node needs to be modified to give permission for the remote node to create the window.

C.2.1 Defining X-Window Output Node

For Unix and DEC-Ultrix machines, a sample command would be:

% **setenv DISPLAY sxt2:0**
% **setenv DISPLAY 192.68.162.109:0**

For DEC-Ultrix machines, it is possible to use DECNET protocol to create the local X-window. This is not recommeded since it is slow, but if it is necessary, a sample command is:

% **setenv DISPLAY sag::0**
% **setenv DISPLAY 24707::0**

For VMS machines, a sample command would be:

$ **set display/create/super/node=sxt2**
$ **set display/create/super/node=sxt2/transport=tcpip**

It is possible to issue the "setenv DISPLAY" command from within IDL by using the routine SETDISP (if the user forgets prior to starting IDL). A sample call would be:

IDL> **setdisp, '192.68.162.109:0'**

C.2.2 Modifying Security to Accept Remote X-Windows

For Unix machines, the following command will temporarily allow all remote nodes permission to create X-windows on your local machine. The command on the second and third lines will only allow access from the node sxt3.

% **xhost +**
% **xhost +sxt3**
% **xhost +sxt3.space.lockheed.com**

It is possible to make a permanent change to the security list of nodes which should be able to create X-windows on your local machine. For the DEC Ultrix or VMS machines using DEC Windows, the following steps are required on the local machine:

1. Find the 'Session Manager' window and make that the current window
2. Click and hold on the 'Customize' item
3. Drag and select the 'Security' item (by releasing the button)
4. Enter the remote host name. An example would be 'sxt2' (without the quotes) if sxt2 was defined in the /etc/hosts file. If not, you would enter the full name 'sxt2.spac.lockheed.com'
5. Click on 'Add' and then 'OK' to exit

When trying to establish a connection between a DEC and a Sun machine, it appears that it might be necessary to have the remote node name and number located in the `/etc/hosts` file. If you have problems after following the instructions above, you should try editing the `/etc/hosts` file and adding the remote host name.

On all Unix machines, the superuser can edit the file `/etc/X0.hosts` and add the names of the nodes which should be allowed access (one machine name per line).

D How to Contribute Software to the Yohkoh System

We encourage users to contribute SW to the system, however we strongly recommend that people at remote sites do not make changes or additions to the directories under $ys. If additions must be made, we recommend that they be placed in the $ys/site/soft branch of the Yohkoh software tree. Software in the $ys/site/soft branch are not changed during software installations and upgrades. The routines placed in the $ys/site/soft branch should not be modified versions of "official" routines since updates to those routines will not be accessed. See the discussion of IDL_PATH to understand how you can have both Yohkoh Software directories and personal directories accessible at the same time.

We have a system in place to facilitate contributions via anonymous FTP from remote systems. A set of Unix script files is used to automate the transfer (all FTP commands are handled within the scripts) and to help the Yohkoh SW managers to handle the contributions in a systematic fashion [5].

There are several categories of user contributions which we are interested in having remote sites submit to the system:

1. Software: Software contributions are expected to follow the guidelines described in appendix of the Reference Guide.
    - Corrections to distributed Yohkoh SW. This category would include any changes which were required at a remote site to fix a known problem (bug) with a distributed routine. It also includes modifications which were required to get the routine to work properly at the particular site due to differences in environment (operating system, shell environment, etc)
    - Enhancements to distributed SW. This includes any changes made at remote sites to add new functions or features and thought to be of a more general interest.
    - New software which would be of use to others and possibly avoid duplication of SW effort. General purpose tools and utilities
2. Non-Software:
    - Non-SW information of general interest. Examples would be hints, clever tricks and mistakes encountered during the use of the Yohkoh SW system. As these might get automatically included in a newsletter without editing, please keep these brief and to the point.

Submissions may be made through the use of the one line command [6]:

% **ys_contrib files [README] [/ysmod]**

[5] The method used in the past was to use the *online* script. We request that people start to use the *ys_contrib* script as the *online* script will be removed at some point in the future.

[6] Requires Yohkoh SW Version 931108.02 or higher

```
Where: files   = SW source, tar or compressed tar file
       README  = optional special announcement or description (file)
       /ysmod  = flag modifications to existing/distributed Yohkoh SW
```

Some examples are:

% **ys_contrib stepper.pro /ysmod**
% **ys_contrib spec_fitter.pro spec_filter2.pro**
% **ys_contrib HINTS**
% **ys_contrib local_lib.tar**

On remote machines, this command (Unix alias) will execute a set of script files which

- connects to anonymous FTP account on the master host
- create subdirectory pub/swmaint/contrib/*node*/*user*
- "FTP puts" the remote files
- sends mail to software account and *user*@*node* announcing new contribution

On ISAS local machines, it checks to see if the person has a /ys/ucon/soft directory and copies it there if it exists. If not, it follows the steps shown above for remote machines.

A cron job running on the SW master will monitor the activity of the user contribution area. It would perform any decompression and tar extraction required (.Z and .tar files) and sends mail to the Yohkoh SW managers if contributions sit around too long without access. Possible disposition of contributed items include:

- move to $ys (atest or ucon) area as appropriate
- delegate review and modification, then mv to $ys
- reject/delete
- e-mail to *user*@*node* notifying

Contributed File Name Conventions - the following restrictions apply.

```
File Type               Required Name

tar                     <name>.tar
compressed              <name>.Z
compressed-tar          <name>.tar.Z
idl source code         <routine>.pro
```

If <routine> is a correction or an enhancement of an existing routine, then the file name must be identical to the existing routine which it corrects or modifies. If filename is HINTS, it is assumed to be "non-software" and may get included in newsletter verbatim.

E Guidelines for Writing Distributed Software

E.1 Names of Routines

The first concern is not to use a name that is already being used. To check if the name exists on the /ys tree on an Ultrix machine, you can use the symbol 'chk_name'. Here is an example (note that the .PRO portion is required):

% **chk_name yodat.pro**

On a VMS machine, the following command should work:

$ **dir ys:[...]yodat.pro**

It is possible to do the search from within IDL on any of the machines using a command like:

IDL> **print, path_lib('yodat')**
IDL> **print, path_lib('yo*')** ;for all routines beginning with 'yo'
IDL> **pr_path_lib, 'yo*'**

The second concern is use care when chosing the name. It should not be general enough to cause conflicts or confusion between all of the other instrument teams. For example, the original name of SXT_DECOMP was DECOMP. Every instrument has a decompression routine, so the name of the instrument needs to be incorporated in the name of the routine for general routines. The name of the routine should not be shorter than four letters and less than 14 characters (so that IDL can compile it properly). Underscores are acceptable, and the file name should be all lower case.

The GT_ routines are a unique set of routines meant solely for the extraction of data out of a Yohkoh data structure. Normally, routines should have the name GET_ unless it is designed to extract data from one fo the Yohkoh data structures.

E.2 Calling Sequence

It is the desire to have the calling sequence be **index,data** for all routines that require the index and data. Please try to avoid having any calling sequence of the form *data,index*.

In general, positional parameters should be used for any input which is required for the procedure. Keyword parameters should be used for the optional parameters.

E.3 Standard Headers

The following standard IDL should be included at the top of each routine. It can be found in the file $DIR_GEN_DOC/idl_template. Every line between ";+" and ";-" will be printed to the screen by the DOC_LIBRARY utility.

```
;+
; NAME:
; PURPOSE:
; CATEGORY:
; CALLING SEQUENCE:
; INPUTS:
; OPTIONAL INPUT PARAMETERS:
; OUTPUTS:
; OPTIONAL OUTPUT PARAMETERS:
; COMMON BLOCKS:
; SIDE EFFECTS:
; RESTRICTIONS:
; PROCEDURE:
; MODIFICATION HISTORY:
;-
```

E.4 Modularize

Instead of making large main programs, try to break the program up into smaller general purpose procedures or functions. Other users will be able to use your routines as building blocks, avoids duplication, and it allows for single point maintenance.

E.5 Portability

It is important to have all software be portable between different computers and different operating systems. One of the biggest mistakes is use of full file names with directories. Since the software needs to work on VMS and Unix machines, use the routine CONCAT_DIR, to concatenate a logical directory name and a file name. For example, the command

IDL> **outfil = concat_dir('$DIR_GEN_OBS', 'obs92_40a.01')**

returns '$DIR_GEN_OBS/obs92_40a.01' and '$DIR_GEN_OBS:obs92_40a.01' for VMS and Unix machines, respectively. Always use logicals pointing to files in the YS directory area; do not write software that points to personal files since those may not be available at other sites.

E.6 Don't Hardwire to Fixed Units

Use /GET_LUN on the file open statements, and /FREE on the window commands

IDL> **openr, lun, filename, /get_lun** ;GOOD EXAMPLE
IDL> **openr, 1, filename** ;BAD EXAMPLE

F Basics for Using FTP

The following are some of the basic commands for using FTP. Several systems have anonymous FTP available. In these cases, use **anonymous** as the username, and your E-mail address for the password.

binary use binary file type for transfer (for files such as idlsave files, Yohkoh data files, ...)

prompt toggle between interactive and non-interactive prompting (it is important to be non-interactive when using 'mget')

cd change default directory on the remote machine

lcd change default directory on the local machine

get copy one file from remote to local

mget copy several files from remote to local

put copy one file from local to remote

A sample session might look like:

% **ftp 133.74.8.100**
username: **guest**
passwd: **your_password**
FTP> **binary**
FTP> **cd /ys/sxt/doc**
FTP> **get travel_sxt.txt**
FTP> **cd /ys/gen/doc**
FTP> **prompt**
FTP> **mget papers.***
FTP> **quit**
%