

Hi,

this is not a trivial task. It begins with reading the data from the HDF file. In case it's HDF5, I have attached a small routine, READ_HDF, which can help you with that task. Once you have extracted your data and longitude and latitude arrays, you want to display them in a chosen projection. The classical routine for this task is IMAGEMAP by Liam Gumley:

<http://cimss.ssec.wisc.edu/~gumley/imagemap.html>

However, I once wrote my own routine based on IMAGEMAP (and a lot of ideas from Rene Preusker and Chris O'Dell) which gives you an object, named, well, MAP_IMAGE, which has the same features as IMAGEMAP and some more. It adds e.g. a colour bar (using David Fannings COLORBAR routine) and lets you choose any of the IDL built-in colour tables plus some more, including one for elevation data and one with blue colours for negative values and red colours for positive values, which is nice for difference plots. You can also choose to use a discrete colour bar. It also allows you to add multiple satellite overpasses or granules subsequently into the same projection. And, above all, it can create plots with black colour onto a white background! :-) The complete file is also attached below.

All keywords accepted by MAP_SET are passed through via _EXTRA, e.g. LONDEL, LATDEL, or the projection types STEREO, SINUSOIDAL, etc.

The file comes with some descriptions on top and some examples at the end, so I would be pleased if you gave it a try. Besides COLORBAR, it also uses a little routine WIN, which is an extension to the built-in WINDOW command. WIN just checks whether a chosen window already exists, thus avoiding destroying and creating the same window over and over again. Once you arranged the windows on the screen, they stay where they are. The file is also attached below.

Now, assuming that the data is in HDF5 and that the appropriate datasets are named "data", "longitude" and "latitude", an IDL code fragment would look like this:

```
<IDL>
data = read_hdf(hdf_file, "data")
lon = read_hdf(hdf_file, "longitude")
lat = read_hdf(hdf_file, "latitude")
m=obj_new("map_image", data, lat, lon)
```

```
obj_destroy, m
</IDL>
```

Now, this is a boring-looking black and white colour bar, so let's try

```
<IDL>
m=obj_new("map_image", data, lat, lon, /rainbow)
obj_destroy, m
</IDL>
```

Which at least adds some colours. If you now set

```
<IDL>
m=obj_new("map_image", data, lat, lon, /rainbow, /sinusoidal)
obj_destroy, m
</IDL>
```

you get a different projection.

```
<IDL>
m=obj_new("map_image", data, lat, lon, /rainbow, /sinusoidal, $
    /horizontal)
obj_destroy, m
</IDL>
```

gives you a horizontal colour bar.

Now you might want to use the TITLE keyword, and REVERT_BACKGROUND for black on white, or CTABLE for a different built-in colour bar, etc. ...

Best regards,

Peter

P.S.: After plotting the images to the screen, you might want to produce jpg or png files. For that task I'd again suggest Liam Gumley and his SAVEIMAGE routine:
<http://cimss.ssec.wisc.edu/~gumley/imagetools.html>

P.P.S.: I guess that many of the lines in the files are too long and will be cropped in this thread, so I'll try to find some webspace where I can put them. I'll let you know when that happened.

```
#####
# read_hdf.pro
#####
```

```

; $Id: read_hdf.pro,v 1.00 2004/06/16
;
; +
; NAME:
;   READ_HDF
;
; PURPOSE:
;   Read one dataset and attributes from a hdf-file
;
; INPUT:
;   filename:   HDF-filename
;   dataset_name: Name of the HDF dataset
;
; KEYWORDS
;   attributes: If set to a named variables, it will return the
attributes
;               associated with the dataset
;   scale:      If set, the data will be scaled with the appropriate
gain
;               and offsets, if available in the data. Existing nodata
values
;               will also be scaled
;   list:       If set, and if dataset_name is actually the name of a
valid
;               dataset group, a list of all datasets and subgroups
of this group
;               is returned. If no dataset_name is set, all datasets
in the parent
;               group "/" are returned.
;               If LIST is set to 1 (equivalent to /LIST) the
complete names
;               are returned, e.g.
;               "/Group1/Subgroup2/Dataset1"
;               "/Group1/Subgroup2/Dataset2"
;               "/Group1/Subgroup2/Dataset3".
;               If LIST is set to a value greater than 1, only the
lowermost level
;               of the all names is returned, e.g.
;               "Dataset1"
;               "Dataset2"
;               "Dataset3".
;   verbose:    Let read_hdf be a bit more talkative about what it
does and
;               tried to do.
;               Setting verbose to 1 will also include the latest
error message (if any), setting verbose to 2 will
;               include all error messages (if any)

```

```

;
;
;
; PROCEDURES:
;   needs h5o_is_group.pro
;
;
;
; MODIFICATION HISTORY:
;   written by: Peter Albert, 16.06.2004
;   Added option for choosing dataset from "/" group members (PA
20050804)
;   Added keyword LIST (PA 20050816)
;   Forgot to close file when LIST was set. Fixed. (PA 20050822)
;   Choosing a group instead of a dataset now leads to recursive
calls
;   to read_hdf until a valid dataset is chosen. If the keyword
;   LIST is set, the list of all subgroups and datasets is
returned.
;   This worked previously just with "/" as base group. (PA
20051004)
;   Included printing of error messages when VERBOSE is set (PA
20051005)
;
;
; -
; =====

```

```

function read_hdf, filename, dataset_name, $
    list = list, $
    attributes = attributes, $
    scale = scale, $
    verbose = verbose

```

```

; Catch errors from e.g. non-existing datasets

```

```

catch, error
if error ne 0 then begin
    if size(file_id, /type) ne 0 then begin
        h5f_close, file_id
        if keyword_set(verbose) then begin
            catch, /cancel
            help, /Last_Message, Output = qwertz_theErrorMessage
            if verbose eq 1 then $
                for j = 0, 1 do $
                    print, "[read_hdf: Error]" +
qwertz_theErrorMessage[j] $
            else $
                for j=0,n_elements(qwertz_theErrorMessage) - 1 do $
                    print, "[read_hdf: Error]" +
qwertz_theErrorMessage[j]

```

```

        endif
    endif
    return, -1
endif

; -----

; If no parameter is given and no keyword is set,
; display the list of datasets in the "root directory"
; of the HDF file. The dataset is thus "/".

if n_params() eq 1 then dataset_name = "/"

file_id = h5f_open(filename)

; Check whether the dataset_name given is already a dataset
; or rather a group of datasets.

n_data = h5o_is_group(file_id, dataset_name) $
    ? h5g_get_nmembers(file_id, dataset_name) $
    : 0

; -----

; If dataset_name is a group of datasets, show either a list of
; datasets (or subgroups) contained within or return
; the list.

if n_data gt 0 then begin

; Concatenate base group name and its contents.
; Make sure that there are not too many "/"s

    sub_dataset_name = strarr(n_data)
    prefix = $
    ( $
        strmid(dataset_name, 0, 1) eq "/" $
        ? dataset_name $
        : "/" + dataset_name $
    ) + $
    ( $
        strmid(dataset_name, strlen(dataset_name)-1, 1) eq "/" $
        ? "" $
        : "/" $
    )

    for i = 0, n_data - 1 do $
        sub_dataset_name[i] = prefix + h5g_get_member_name(file_id,

```

```

dataset_name, i)

    h5f_close, file_id

; If keyword LIST is set, just return the list of subgroups and
datasets

    if keyword_set(list) then return, list eq 1 $
        ? sub_dataset_name $
        : strmid(sub_dataset_name, strpos(sub_dataset_name[0], "/"),
/reverse_search) + 1) $
    else begin

; If keyword LIST is not set, show the list of subgroups and datasets,
; ask for selection and call read_hdf again with the chosen
; subgroup or dataset

        for i = 0, n_data - 1 do $
            print, i, sub_dataset_name[i], format = '(i3, ": ", a)'

        s = ""
        read, s, prompt = "Read dataset number: "

; Recursively call read_hdf with new selected dataset name

        out = read_hdf( $
            filename, $
            sub_dataset_name[fix(s)], $
            list = list, $
            attributes = attributes, $
            scale = scale, $
            verbose = verbose $
        )
        return, out
    endelse
endif

; -----

; Now that we have a valid dataset name, we can go ahead
; and read it from the file

    if keyword_set(verbose) then print, "[INFO read_hdf:] Reading " +
dataset_name
    dataset_id = h5d_open(file_id, dataset_name)
    dataset = h5d_read(dataset_id)

```

```

; Read the attributes

if (n_elements(attributes) ne 0) then tempvar =
size(temporary(attributes))
n_att = h5a_get_num_attrs(dataset_id)
if n_att gt 0 then begin
  for j = 0, n_att - 1 do begin
    att_id = h5a_open_idx(dataset_id, j)
    name = h5a_get_name(att_id)
    att_data = h5a_read(att_id)
    if strpos(strlower(name), "no_data") ne -1 then att_data =
float(att_data)
    attributes = j ne 0 ? $
      create_struct(attributes, name, att_data) : $
      create_struct(name, att_data)
    h5a_close, att_id
  endfor
endif
h5d_close, dataset_id

; --- < Identify nodata value > ---
; --- Sometimes it's NODATA, sometimes NO_DATA ---

if size(attributes, /type) eq 8 then begin
  tags = strlower(tag_names(attributes))
  index = where(strpos(tags, "nodata") ne -1)
  if index[0] eq -1 then index = where(strpos(tags, "no_data") ne
-1)
  attributes = create_struct(attributes, "nodata_id", index[0])
endif

; --- < /Identify nodata value > ---

; --- < Scale data > ---

if keyword_set(scale) then begin
  gain = 1.0
  offset = 0.0

  if size(attributes, /type) eq 8 then begin
    tnames = strlower(tag_names(attributes))
    i = where(tnames eq "gain", c)
    if c eq 1 then gain = attributes.(i[0])
    i = where(tnames eq "scaling_factor", c)
    if c eq 1 then gain = attributes.(i[0])
    i = where(tnames eq "intercept", c)
    if c eq 1 then offset = attributes.(i[0])
    i = where(tnames eq "offset", c)

```

```

        if c eq 1 then offset = attributes.(i[0])
    endif
    if gain ne 1 or offset ne 0 then begin
        if keyword_set(verbose) then print, "[INFO read_hdf:] Scaling
" + dataset_name
        dataset = float(dataset) * gain + offset
        if attributes.nodata_id ne -1 then begin
            if keyword_set(verbose) then print, "[INFO read_hdf:]
Scaling " + dataset_name + "." + tags[attributes.nodata_id]
            attributes.(attributes.nodata_id) =
attributes.(attributes.nodata_id) * gain + offset
        endif
    endif
endif
endif

```

```

; --- < /Scale data > ---

```

```

    h5f_close, file_id
    return, dataset
end

```

```

#####
# map_image__define.pro
#####

```

```

;$Id: map_image__define.pro,v 1.9 2005/04/26
;
;
;+
; NAME:
;   map_image
;
; PURPOSE:
;   Colour coded display of georeferences data
;
; BASIS:
;   This program is a mixture of Liam Gumley's IMAGEMAP, Rene
Preusker's
;   FUB_IMAGE and Chris O'Dell's WIS_IMAGE. They all use MAP_SET,
;   CONVERT_COORD and and TV. What's new is that this one is object
oriented,
;   allowing the easy overlay of different images and the zooming into
the images.
;   Which does not necessarily requires object oriented programs, but
anyway.
;   Uses COLORBAR (by David Fanning) to draw the colorbar.

```



```

;
; CALLING SEQUENCE:
;
;   THERE ARE A LOT OF EXAMPLES AT THE END OF THE PROGRAM!
;   CALL MAP_IMAGE_DEMO
;
;   m = obj_new("map_image", image, lat, lon)
;   obj_destroy, m
;
;   OR, after compilation of map_image__define.pro
;
;   map_image, image, lat, lon
;
; INPUTS:
;   IMAGE: 1- or 2-dimensional data field. Or (x, y, 3) or (n, 3)
;   data field for true colour images.
; LAT:
;   Array or vector of latitudes
; LON:
;   Array or vector of longitudes
;
; OPTIONAL KEYWORDS:
;   Generally, all keywords related to MAP_SET, MAP_GRID, etc. can be
passed
;   through via the _EXTRA keyword and are not all repeated here.
;   Most important may be the projection keywords like STEREO,
SINUSOIDAL
;   and keywords like LONDEL, LATDEL, LIMIT or BOX_AXES.
;
;   Keywords specific to MAP_IMAGE_DEFINE are:
;
; NO_COPY:
;   Saves memory when set, but the original
;   arrays are lost for the calling routine.
;
; NO_PROJECT:
;   Suppresses image drawing directly when object is created.
;   This is useful when you e.g. want to modify the colour table
;   prior to projecting the data.
;
; NO_DRAW:
;   Only suppresses drawing of the projected data. Useful when
plotting
;   several datasets subsequently in the same projection.
;
; NO_ERASE:
;   If set, the plot region is not erased. Necessary when plotting
;   several datasets subsequently in the same window. See examples

```

below.

```
;
; NO_SCALE:
;   If set, the original data is not bytescaled.
;
; POLON, POLAT, ROT:
;   Must be set as keywords in order to get passed through to
map_set.
;
; BOTTOM and NCOLOR:
;   Are handled like in Dave Fannings COLORBAR.PRO
;
; WINDOW:
;   Set plot window
;
; REVERT_BACKGROUND:
;   If set, the back- and foreground colors are switched,
;   typically useful for creating black on white plots
;   instead of the standard black background.
;
; CTABLE:
;   Set color table
;
; RAINBOW:
;   Use a predefined, colour enhanced rainbow colour table
;
; GREYSCALE:
;   Use greyscale colour table.
;
; ELEVATION
;   Use a 'elevation' type of color table, ranging from blue over
green and
;   brown to white
;   Attention: Currently, only 79 colours are defined, so a call
should be
;   combined with ", ncolors=79"
;
; BWR
;   Use a 'blue white red' colour table with negative values in blue,
0 in white
;   and positive values in red
;
; DISCRETE:
;   Use discrete colours.
;
; RED, GREEN, BLUE:
;   Specify red, green and blue values for discrete colours.
;   N.B.: The number of elements of the RED, GREEN and BLUE
```

```

; vectors must equal the number of elements of the DISCRETE
; vector minus 1 (Assume DISCRETE set to [0, 10, 20], then 2 colours
; are needed for the range 0 to 10 and 10 to 20).
;
; FLIP_COLOURS
; If set, the colour bar is reverted, eg. is ranging from red to
blue instead
; of blue to red
;
; MAGNIFY:
; If plotting low resolution data, this keyword basically draws
borders
; around each pixel in order to fill holes. Can be set to scalar
values greater
; than zero.
;
; HORIZONTAL:
; If set, a horizontal colour bar is plotted. Default is vertical.
;
; DEBUG:
; Print some debug information.
;
;-----;
;
; OBJECT METHODS:
;
; MAP_IMAGE::COLORS
; Set color table
; KEYWORDS:
; RAINBOW, GREYSCALE, CTABLE, DISCRETE
;
; MAP_IMAGE::WORLDVIEW
; Copied from the GEOV.PRO, provided by Andrey Savtchenko of the
MODIS
; Data Support Team, this method shows the location of the data on a
globe.
;
; MAP_IMAGE::project
; Performs the actual projection of the data on the desired grid.
; KEYWORDS:
; NO_DRAW, NO_ERASE
;
; MAP_IMAGE::display
; Displays the projected data.
;
; MAP_IMAGE::rdpix
; Executes RDPPIX on the projected dataset
;

```

```

; MAP_IMAGE::colorbar
;   This method calls David Fannings COLORBAR.pro. Different to
COLORBAR, the
;   default colour bar is vertical. You can set a keyword HORIZONTAL
on order to get
;   a horizontal one.
;
;
; MAP_IMAGE::draw_border
;   Just draw a line around the draw area as this one sometimes gets
lost ...
;
;
; MAP_IMAGE::zoom
;   Call this method in order to interactively zoom into an image.
;
;
; MAP_IMAGE::init
;   Creates a new object.
;   KEYWORDS:
;     NO_COPY, NO_PROJECT, DEBUG
;
;
; MAP_IMAGE::cleanup
;   Dereference pointers etc.
;
;
; MAP_IMAGE::message
;   Internal routine for debug information
;
;
; MAP_IMAGE::remove_tag
;   Internal routine for the handling of structures
;
;
; MAP_IMAGE::set_range
;   Internal routine which does a first guess for nicely looking
values for the colour bar.
;
;
; MAP_IMAGE::set_limit
;   Internal routine which sets a first-guess limit for the map
projection.
;
;
; MAP_IMAGE::decomposed
;   Sets DEVICE, decomposed = 0 for 8 bit images and decomposed = 1
for true colour images
;
;
; MAP_IMAGE::limit428
;   map_image expects the LIMIT keyword to MAP_SET to be an 8-element
vector. This routines
;   converts a given 4-element vector into an 8-element vector.
;
;
; MAP_IMAGE::set_one_var
;   Internal method for structure handling. All keywords passed to
map_image via _EXTRA

```

```

; are stored in an internal structure.
;
; MAP_IMAGE::set_var
; Internal method for structure handling. All keywords passed to
map_image via _EXTRA
; are stored in an internal structure.
;
; MAP_IMAGE::get_var
; Return values of internal structure elements.
;
; MAP_IMAGE::debug
; Can be used for debug purposes in order to stop within a method
(and get access to
; object variables).

;-----;
; EXTERNAL SUBROUTINES
; win.pro
; colorbar.pro
;
;-----;
; CAVEATS:
; MAP_IMAGE works on 8 bit colour mode, i.e. you should set
; DEVICE, PSEUDE=8 after you started IDL!
;
;-----;
; -
;
; Modification history:
; Created by Peter Albert, 23.9.2004
;
; 14.10.2004 PA map_image::colors now checking for keyword
"discrete" after all other
; keywords -> discrete works
together with "ctable"
; 01.12.2004 PA map_image::colors now sets self.var.draw.maxi if
max(image) or the value
; provided via "MAXI= " is lower
than max(discrete)
; It also checks ranges when the
colour index (idx) is
; set.
; 13.12.2004 PA map_image::init and ::cleanup: now restoring
original values
; of !p.color and !p.background
;
; The background keyword to

```

```

dilate does not what I think, so
;           magnifying the image plotting
black on white (revert_background)
;           did not work. I am now setting
all pixels to 0 at the start (::project)
;           and do not use the background
keyword with dilate (::display). The
;           remaining black pixels are set
to !p.background afterwards.
; 28.12.2004  PA  map_image::project: Consistent use of
self.var.draw.xoffset instead of corners[0] should
;           now avoid the fact that
sometimes the map_set frame was erased by the image.
;           Additionally, ceil and floor is
now used when calculating size and offset
;           from normalized device
coordinates.
;
; 29.12.2004  PA  colorbar / ex12: The new example ex12 shows how
data with logarithmic scaling
;           can be displayed. Attention:
The subroutine colorbar.pro
;           was modified!
;
;
; 05.01.2005  PA  colorbar / ex12: The updated example 12
illustrates how logarithmic scaling can be achieved
;           without modifications of
colorbar.pro.
;           map_image::project: There was a confusing offset
of 2 at the numbers of used colors when
;           bytescaling the data. Removed.
;
;
; 05.01.2005  PA/JS map_image::colors: Added colour table "maxcolors"
;
; 10.01.2005  PA  map_image::colors: When keywords discrete and red
are set, only set r[0] = 0 (and g, b)
;
;
; 11.01.2005  PA  map_image::project / map_image::colors: Going
back to saving 2 colours for background and void-color and 1 for
;           the foreground colour. Using a
maximum of 253 colours for the image.
;           Avoid error messages when
displaying byte data with a negative user-defined minimum value
;           map_image:draw_border: Added routine
;
;
; 12.01.2005  PA  map_image::project: Corected small bug with
respect to size and offset of plot region
;           map_image__define: Changed type of

```

```

self.var.draw.color / background from byte to long
;           map_image::project: No bytscl when displaying
true colour images
;           map_image::display: No setting of background
colour when displaying true colour images
;
; 13.01.2005  PA  Corrected some bugs with regard to true colour
images, magnify and postscript output
;
; 24.01.2005  PA  map_image::colors: Renamed keyword "maxcolors" to
"extended_rainbow" in order to avoid
;           conflicts with
self.var.draw.maxi (When keyword "MAX) was set
;           it was recognised for
self.var.draw.maxi and for "maxcolors"
;
; 27.01.2005  PA  map_image::colors: With discrete colours, the
colours were sometimes defined such that
;           colorbar.pro sometimes showed
the steps at a slightly wrong position.
;           Workaround with setting the
number of colours.
;           No revert_background for
postscript
;
; 28.01.2005  PA  map_image::set_one_var: Call set_range for new
image only when mini and maxi are not yet set
;           map_image::remove_tag: The new structure was
built from *self.var.internal._extra instead from struct_in
;
; 01.02.2005  PA  map_image::init, map_image::colorbar: When called
with keyword "horizontal=0", a
;           horizontal colourbar was drawn
anyway. Fixed
;           map_init::project: Added keyword no_scale
(controlled by self.var.draw.no_scale)
;
; 11.02.2005  PA  map_image::colors: Added keyword flip_colours
;
; 24.02.2005  PA  map_init::colours: Added keyword "elevation"
;
; 01.03.2005  PA  Added map_image::rdpix
;
; 03.03.2005  PA  map_image::set_limit: Improved calculation of
default limits
;           for map_set
;           map_image::colors: Is it a bug or is it a
feature? When called for the first time

```

```

;           TVLCT modifies !p.color. It
will now be set back to the original
;           value.
; 17.03.2005  PA  map_init::colours: Added keyword "bwr"
; 15.04.2005  PA  map_init::project: Check for NaNs from
convert_coord before proceeding
; 26.04.2005  PA  map_init::project: When drawing true colour
images using /REVERT, areas where all three
;           channels are equal to zero
were drawn in white. Now, the minimum value
;           for all three channels is set
to one.
; 19.07.2005  PA  map_init::project/display: Added _extra = _extra
to calls to win. Allows setting XSIZE and
;           YSIZE in calls to map_image
; 12.08.2005  PA  map_image::colors: When keyword BWR was set
together with NCOLORS and / or BOTTOM,
;           white was not at the 0 value
;           map_image::project: When the keyword
REVERT_BACKGROUND was abbreviated to REV, it was interpreted
;           as REVERSE by the first call
to map_set, and the image was flipped around the
;           x-axis
; 18.08.2005  PA  map_image::display: deleted /horiz from call to
map_grid
; 02.09.2005  PA  map_image::worldview: Crashed when lon / lat data
was
;           1D. Not fixed but no execution
in
;           this case.
; 25.10.2005  PA  map_image::get_var: Added function
;-----
;-----
;
=====
=====
; Set predefined colors
;
=====
=====

pro map_image::colors, $
    rainbow = rainbow, $
    extended_rainbow = extended_rainbow, $
    greyscale = greyscale, $
    elevation = elevation, $
    bwr = bwr, $

```



```

        ctable = ctable, $
        discrete = discrete, $
        red = red, green = green, blue = blue, $
        revert_background = revert_background, $
        flip_colours = flip_colours

    if keyword_set(rainbow) then begin
        self -> message, "[map_image::colors]: Rainbow color table"
        r = interpol([ 0, 0, 0, 0, 0, 200, 255, 255, 255, 180,
100], indgen(11), findgen(254) / 253 * 10)
        g = interpol([ 0, 0, 125, 255, 255, 255, 255, 255, 125, 0, 0,
0], indgen(11), findgen(254) / 253 * 10)
        b = interpol([125, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0,
0], indgen(11), findgen(254) / 253 * 10)
        r = [0, r, 255] & g = [0, g, 255] & b = [0, b, 255]
        color_save = !p.color & tvlct, r, g, b & !p.color = color_save
    endif

    if keyword_set(extended_rainbow) then begin
        self -> message, "[map_image::colors]: Extended Rainbow color
table"
        r = interpol([152, 217, 105, 0, 0, 7, 109, 210, 250, 250,
250, 221, 156, 100, 80], indgen(15), findgen(254) / 253 * 14)
        g = interpol([ 0, 0, 0, 125, 250, 245, 224, 250, 216, 159,
101, 46, 0, 0, 34], indgen(15), findgen(254) / 253 * 14)
        b = interpol([ 91, 230, 244, 250, 250, 140, 31, 0, 0, 0,
0, 46, 0, 0, 40], indgen(15), findgen(254) / 253 * 14)
        r = [0, r, 255] & g = [0, g, 255] & b = [0, b, 255]
        color_save = !p.color & tvlct, r, g, b & !p.color = color_save
    endif

    if keyword_set(elevation) then begin
        self -> message, "[map_image::colors]: 'Elevation' style: blue,
green, yellow, brown, white"

        r = bytarr(256)
        g = bytarr(256)
        b = bytarr(256)
        r[2] = 0 & g[2] = 0 & b[2] = 150
        r[3:20] = 0 & g[3:20] = bindgen(18) * (190.-120.) / 17. + 120 &
b[3:20] = 0
        r[21:40] = bindgen(20) * (220.) / 19. & g[21:40] = 190 & b[21:40]
= 0
        r[41:60] = 220 - bindgen(20) * (30) / 19. & g[41:60] = 190 -
bindgen(20) * (80) / 19. & b[41:60] = 0
        r[61:80] = bindgen(20) * 65 / 19. + 190
        g[61:80] = bindgen(20) * 145 / 19. + 110
        b[61:80] = bindgen(20) * 255 / 19.

```

```

r[81] = 235 & g[81] = 235 & b[81] = 235
color_save = !p.color & tvlct, r, g, b & !p.color = color_save

endif

if keyword_set(bwr) then begin
    self -> message, "[map_image::colors]: Blue to red"
    saturation = filtarr(self.var.draw.ncolors)
    value = intarr(self.var.draw.ncolors) + 1
    hue = intarr(self.var.draw.ncolors)

    white_index = round((self.var.draw.ncolors * (-1. *
self.var.draw.mini) / (self.var.draw.maxi - self.var.draw.mini))>0)
    hue [0 : white_index] = 240
    saturation[0 : white_index] = (1. - findgen(white_index+1) /
white_index) * self.var.draw.ncolors / 253.
    saturation[white_index + 1 : self.var.draw.ncolors - 1] =
findgen(self.var.draw.ncolors-white_index-1) /
(self.var.draw.ncolors-white_index-2) * self.var.draw.ncolors / 253.
    color_convert, hue, saturation, value, r, g, b, /hsv_rgb
    rr = intarr(254) & gg = intarr(254) & bb = intarr(254)
    rr[self.var.draw.bottom-1] = r
    gg[self.var.draw.bottom-1] = g
    bb[self.var.draw.bottom-1] = b
    r = [0, rr, 255] & g = [0, gg, 255] & b = [0, bb, 255]
    color_save = !p.color & tvlct, r, g, b & !p.color = color_save
endif

if keyword_set(ctable) then begin
    self -> message, "[map_image::colors]: Color table " +
string(ctable, format = '(i2)')
    loadct, ctable, /silent
endif
if keyword_set(greyscale) then begin
    self -> message, "[map_image::colors]: Greyscale color table"
    loadct, 0, /silent
endif
if keyword_set(discrete) then begin
    self -> message, "[map_image::colors]: Discrete colors"
    if self.var.draw.maxi lt max(discrete) then begin
        self.var.draw.maxi = max(discrete)
        self -> message, "[map_image::colors]: Setting maxValue to "
+ string(self.var.draw.maxi)
    endif

    nnn = n_elements(discrete) - 1

```

```

; Setting number of colors such that colorbar shows the steps at the
correct position
    self.var.draw.ncolors = floor(float(self.var.draw.ncolors) / nnn)
* nnn + 1
;   print, nnn, self.var.draw.ncolors

    ddd = bytscl([min(*self.var.data.image, max=maxi), discrete,
maxi], $
        min = self.var.draw.mini, $
        max = self.var.draw.maxi, $
        top = self.var.draw.ncolors - 1, /nan) +
self.var.draw.bottom
    ddd = ddd[1:nnn+1]
    idx = intarr(256) - 1
    for i = 0, nnn-1 do if ddd[i] lt ddd[i+1] then
idx[ddd[i]:ddd[i+1]] = i
        if keyword_set(red) and keyword_set(blue) and keyword_set(green)
then begin
            if n_elements(red) ne nnn or n_elements(green) ne nnn or
n_elements(blue) ne nnn then begin
                print, nnn, format = '("[map_image::colors] ERROR: red,
green and blue must have " , i3, " elements")'
                help, red, green, blue
                return
            endif
        endif else begin
;Pick the colours from within the
chosen intervals
            tvlct, red, green, blue, /get
            index = round(findgen(nnn) / (nnn - 1.) *
(self.var.draw.ncolors - 1) + self.var.draw.bottom)
            red = red [index]
            green = green[index]
            blue = blue [index]
        endelse
        r = red[idx>0] & g = green[idx>0] & b = blue[idx>0]
        bad = where(idx eq -1)
        if bad[0] ne -1 then begin
            r[bad] = self.var.draw.void_color
            g[bad] = self.var.draw.void_color
            b[bad] = self.var.draw.void_color
        endif
        color_save = !p.color & tvlct, r, g, b & !p.color = color_save

    endif

    if keyword_set(flip_colours) then begin
        tvlct, r, g, b, /get

```

```

    color_save = !p.color & tvlct, reverse(r), reverse(g), reverse(b)
& !p.color = color_save
endif

```

```

;Set white, black and void-color:

```

```

    tvlct, r, g, b, /get
    r[0] = 0 & g[0] = 0 & b[0] = 0
    r[255] = 255 & g[255] = 255 & b[255] = 255
    r[1] = self.var.draw.void_color & g[1] = self.var.draw.void_color &
b[1] = self.var.draw.void_color
    color_save = !p.color & tvlct, r, g, b & !p.color = color_save

```

```

    if keyword_set(revert_background) then begin
        if strupcase(!d.name) eq 'PS' then begin
            self -> message, "[map_image::colors]:Don't revert colours
when using postscript"
        endif else begin
            self -> message, "[map_image::colors]: Reverting background
and plot colour:"
            background = !p.background
            color = !p.color
            self -> message, "[map_image::colors]: !p.background: " + $
                strcompress(string(background), /rem) + " --> " + $
                strcompress(string(color), /rem)
            self -> message, "[map_image::colors]: !p.color: " + $
                strcompress(string(color), /rem) + " --> " + $
                strcompress(string(background), /rem)

            !p.background = color
            !p.color = background
        endelse
    endif

```

```

end

```

```

;
=====
=====
; Debug messages
;
=====
=====

```

```

pro map_image::message, msg_string, time = time

```

```

    if keyword_set(time) then begin
        if self.var.internal.debug then print,

```

```

string(systime(1)-self.var.internal.time, format = '(f5.1, " sec. :
")') + msg_string
endif else begin
    if self.var.internal.debug then print, msg_string
endelse
end

;
;
=====
=====
; Location on the globe
;
;
=====
=====

pro map_image::worldview

; This IDL subroutine was copied from geov.pro, developed at the
; Modis Data Support Team
; by Andrey Savtchenko,(asavtche@daac.gsfc.nasa.gov).

tv!ct, r, g, b, /get
loadct,0, /silent
dims=size(*self.var.data.latitude)
if dims[0] eq 2 then begin
    lon0=(*self.var.data.longitude)[dims[1]/2,dims[2]/2]
    lat0=(*self.var.data.latitude)[dims[1]/2,dims[2]/2]
;latmin((*self.var.data.latitude))+(max((*self.var.data.lati tude))-min((*self.var.data.latitude)))/2.

mpl=[(*self.var.data.latitude)[0,dims[2]-1],(*self.var.data. longitude)[0,dims[2]-1],$
(*self.var.data.latitude)[0,0],(*self.var.data.longitude)[0, 0],$
(*self.var.data.latitude)[dims[1]-1,0],(*self.var.data.longi tude)[dims[1]-1,0],$
(*self.var.data.latitude)[dims[1]-1,dims[2]-1],(*self.var.da ta.longitude)[dims[1]-1,dims[2]-1]]

window,22,xsize=400,ysize=400,retain=2,title="granule geolocation"
map_set,lat0,lon0,/ortho,/isotropic,/noborder,xmargin=0,$
    ymargin=0,/horizon,/grid,color=100, londel = 15, latdel=15

map_continents,fill_continents=1,color=40
map_continents,/coasts,color=100

oplot,(*self.var.data.longitude)[*,0],(*self.var.data.lati tu de)[*,0],psym=3,color=255

```

```

    oplot,(*self.var.data.longitude)[*,dims[2]-1],(*self.var.data.latitude)[*,dims[2]-1],psym=3,color=255

    oplot,(*self.var.data.longitude)[0,*],(*self.var.data.latitude)[0,*],psym=3,color=255

    oplot,(*self.var.data.longitude)[dims[1]-1,*],(*self.var.data.latitude)[dims[1]-1,*],psym=3,color=255
    tvlct, r, g, b
endif else begin
    print, "[map_image::worldview]: Latitude and Longitude data has to
be 2D for this subroutine."
endelse
    return
end

;
=====
; Remove tag from structure
;
=====

function map_image::remove_tag, struct_in, tag

    if size(struct_in, /type) eq 8 then begin
        st_names = strlowercase(tag_names(struct_in))
        for i = 0, n_tags(struct_in) - 1 do begin
            match = -1
            for j = 0, n_elements(tag) - 1 do if strpos(st_names[i],
strlowercase(tag[j])) ne -1 then match = 0
            if match eq -1 then $
                output = size(output, /type) eq 8 $
;                ? create_struct(output, st_names[i],
(*self.var.internal._extra).(i)) $
;                : create_struct(      st_names[i],
(*self.var.internal._extra).(i))
                ? create_struct(output, st_names[i], struct_in.(i)) $
                : create_struct(      st_names[i], struct_in.(i))
            endfor
        endif else output = 0
        return, output
    end

;
=====
; Project image
;

```

```
=====
=====
```

```
pro map_image::project, $
    no_draw = no_draw, $
    no_erase = no_erase, $
    _extra = _extra

    self -> set_var, _extra = _extra
    self -> colors, _extra = _extra
    win, self.var.internal.window, _extra = _extra

; Define map projection if keyword no_erase is not set

    if not keyword_set(no_erase) or (size(*self.var.draw.draw_image))[0]
    eq 0 then begin

        ; First, see whether p0lon, p0lat or
        rot have user-defined values

        if size(*self.var.internal._extra, /type) eq 8 then begin
            p0lon =
            max(strpos(strlowercase(tag_names(*self.var.internal._extra)), 'p0lon'))
            eq -1 $
                ? 0 $
                : (*self.var.internal._extra).p0lon
            p0lat =
            max(strpos(strlowercase(tag_names(*self.var.internal._extra)), 'p0lat'))
            eq -1 $
                ? 0 $
                : (*self.var.internal._extra).p0lat
            rot =
            max(strpos(strlowercase(tag_names(*self.var.internal._extra)), 'rot')) eq
            -1 $
                ? 0 $
                : (*self.var.internal._extra).rot
            endif else begin
                p0lon = 0 & p0lat = 0 & rot = 0
            endelse

            self -> message, /time, string(p0lon, p0lat, rot, format =
            '("[map_image::project]: p0lon, p0lat, rot: ", 3f6.1)')
            map_set, p0lat, p0lon, rot, $
            _extra = self -> remove_tag(*self.var.internal._extra,
            ['tit', 'lond', 'latd', 'rev'])
            endif

; Corners of draw region:
```

```
; x, y, lower right, x, y, upper left in device coordinates
; don't use *self.var.proj.position as things can be confused using
!p.multi
```

```
corners = (!d.x_size * !x.window, !d.y_size * !y.window)[[0, 2, 1,
3]] * self.var.draw.scalef
self.var.draw.xoffset = ceil(corners[0])
self.var.draw.yoffset = ceil(corners[1])
```

```
; Size of the draw region
```

```
xsize = floor(!d.x_size * (!x.window[1] - !x.window[0])) *
self.var.draw.scalef)
ysize = floor(!d.y_size * (!y.window[1] - !y.window[0])) *
self.var.draw.scalef)
```

```
; Create new image or add no data to existing image
if not keyword_set(no_erase) or (size(*self.var.draw.draw_image))[0]
eq 0 then begin
```

```
self -> message, /time, string(xsize, ysize, format =
'("[map_image::project]: Create new image (" , i5, " , " , i5, ")")')
ptr_free, self.var.draw.draw_image
self.var.draw.draw_image = ptr_new(self.var.draw.true_color $
? bytarr(xsize, ysize, 3) $
: bytarr(xsize, ysize))
```

```
endif else self -> message, /time, string(xsize, ysize, format =
'("[map_image::project]: Add to existing image (" , i5, " , " , i5, ")")')
```

```
self -> message, /time, "[map_image::project]: Convert_coord ..."
index = convert_coord(*self.var.data.longitude,
*self.var.data.latitude, /data, /to_device) * self.var.draw.scalef
self -> message, /time, "[map_image::project]: Convert_coord done"
```

```
; Check for pixels where convert_coord produces NaN:
bad = where(finite(index[0, *]) eq 0 or finite(index[1, *]) eq 0, c)
if c gt 0 then begin
index[0, bad] = 0
index[1, bad] = 0
endif
```

```
x = round(0 > (index[0, *] - self.var.draw.xoffset) < (xsize - 1))
y = round(0 > (index[1, *] - self.var.draw.yoffset) < (ysize - 1))
```

```
if c gt 0 then begin
x[bad] = -1
y[bad] = -1
endif
```



```

self -> message, /time, "[map_image::project]: Project image start"
if self.var.draw.true_color then begin
  z = x*0
  case (size(*self.var.data.image))[0] of
    3: for i = 0, 2 do (*self.var.draw.draw_image)[x, y, z * 0 +
i] = (*self.var.data.image)[*, *, i] > 1
    2: for i = 0, 2 do (*self.var.draw.draw_image)[x, y, z * 0 +
i] = (*self.var.data.image)[*, i] > 1
    else: begin
      print, "[map_image::project]: I am confused about the
size of the image... giving up"
      help, *self.var.data.image
    end
  endcase
endif else begin
  ; When displaying discrete byte data it
is sometimes useful to set self.var.draw.mini to -0.5 in order to
  ; have a nice colour bar. In these
cases bytscl creates an error message when applied to the byte data.
  if self.var.draw.no_scale ne 0 then $
    dum_data = *self.var.data.image $
  else $
    if size(*self.var.data.image, /type) eq 1 and
self.var.draw.mini lt 0 then $
      dum_data=bytscl(fix(*self.var.data.image), $
        min = self.var.draw.mini, $
        max = self.var.draw.maxi, $
        top = self.var.draw.ncolors - 1, /nan) +
self.var.draw.bottom $
    else $
      dum_data=bytscl(*self.var.data.image, $
        min = self.var.draw.mini, $
        max = self.var.draw.maxi, $
        top = self.var.draw.ncolors - 1, /nan) +
self.var.draw.bottom

    if (*self.var.draw.void_index)[0] ne -1 then begin
      self -> message, string(self.var.draw.void_color, format =
'("[map_image::project]: Setting void pixels to ", i3)')
      dum_data[( *self.var.draw.void_index)] = 1
    endif

    (*self.var.draw.draw_image)[x, y] = (dum_data)
    ptr_free, self.var.draw.draw_image_orig
    self.var.draw.draw_image_orig =
ptr_new(fltarr((size(*self.var.draw.draw_image))[1:2]))
    (*self.var.draw.draw_image_orig)[x, y] = *self.var.data.image

```

```

endelse
self -> message, /time, "[map_image::project]: Project image done"

if not keyword_set(no_draw) then self -> display, _extra = _extra ;,
location = location
end

;
=====
=====
; Draw projected image
;
=====
=====

pro map_image::display, _extra = _extra

self -> set_var, _extra = _extra

; Magnify image
if self.var.draw.magnify ne 0 then $
    structuring_element = replicate(1, 3, 3)

if self.var.draw.magnify gt 0 then begin
    self -> message, "[map_image::project]: Magnify image"
    for mm = 1, fix(self.var.draw.magnify) do begin
        if strupcase(!d.name) eq 'PS' then begin
            idx = where(*self.var.draw.draw_image eq !p.background)
            if idx[0] ne -1 then (*self.var.draw.draw_image)[idx] = 0
        endif
        *self.var.draw.draw_image = dilate(*self.var.draw.draw_image,
structuring_element, /gray, /constrained)
;;      You don't need the fill and loc stuff when the keyword
"constrained" is set!
;      fill = dilate( *self.var.draw.draw_image, structuring_element,
/gray, background = !p.background)
;      loc = strupcase(!d.name) ne 'PS' $
;      ? where( ( fill ge !p.background ) and (
*self.var.draw.draw_image eq !p.background ), count ) $
;      : where( ( fill le !p.background ) and (
*self.var.draw.draw_image eq !p.background ), count )
;      if count ge 1 then (*self.var.draw.draw_image)[loc] =
fill[loc]
;      print, count
    endfor
endif
if self.var.draw.magnify lt 0 then begin
    count = 1

```

```

while count gt 0 do begin
  self -> message, "[map_image::project]: Auto-Magnify image"
  *self.var.draw.draw_image = dilate(
*self.var.draw.draw_image, structuring_element, /gray, background = (1
- self.var.draw.true_color) * !p.background, /constrained)
; fill = dilate( *self.var.draw.draw_image, structuring_element,
/gray, background = !p.background)
; loc = strupcase(!d.name) ne 'PS' $
; ? where( ( fill ge !p.background ) and (
*self.var.draw.draw_image eq !p.background ), count ) $
; : where( ( fill le !p.background ) and (
*self.var.draw.draw_image eq !p.background ), count )
; if count gt 0 then (*self.var.draw.draw_image)[loc] =
fill[loc]
endwhile
endif

```

```

; The image was originally set to 0, as I am not able to let the
BACKGROUND
; keyword of dilate do what I expect ... so here we set all black
pixels to

```

```

; !p.background
if self.var.draw.true_color and !p.color eq 0 then begin
  idx = where(total(*self.var.draw.draw_image, 3) eq 0)
  if idx[0] ne -1 then begin
    nnn = (size(*self.var.draw.draw_image))[5] / 3
    (*self.var.draw.draw_image)[[idx, idx + nnn, idx + nnn * 2]]
= 255
  endif
endif else begin
  idx = where(*self.var.draw.draw_image eq 0)
  if idx[0] ne -1 then (*self.var.draw.draw_image)[idx] =
!p.background
endelse
self.var.draw.magnify = 0
win, self.var.internal.window, _extra = _extra
ss = size(*self.var.draw.draw_image) / self.var.draw.scalef
tv, *self.var.draw.draw_image, $
  self.var.draw.xoffset / self.var.draw.scalef, $
  self.var.draw.yoffset / self.var.draw.scalef, $
  xsize = ss[1], ysize = ss[2], $
  true = self.var.draw.true_color * 3
map_continents, /hires, /continents, _extra =
*self.var.internal._extra ;, col = !p.background
map_grid, _extra = *self.var.internal._extra

if not self.var.draw.no_color_bar then self -> colorbar
self -> draw_border

```

```

end

;
=====
=====
; Use RDPIX
;
=====
=====

pro map_image::rdpix

win, self.var.internal.window
rdpix, *self.var.draw.draw_image_orig, $
    self.var.draw.xoffset / self.var.draw.scalef, $
    self.var.draw.yoffset / self.var.draw.scalef
end

;
=====
=====
; Draw color bar
;
=====
=====

pro map_image::colorbar, _extra = _extra

    self -> set_var, _extra = _extra

; self -> remove_tag:
; Create temporary _extra structure with all elements of
*self.var.internal._extra but position.
; Position must not be passed to colorbar, as it controls the position
of the mapped image
; The position of the colorbar is controlled by
self.var.draw.cb_position.

; Now this is difficult :-)
; The default behaviour of 'colorbar' is to draw a horizontal colorbar.
I want the default to
; be vertical, therefore I am checking whether a keyword like
'horizontal' is provided in
; _extra, otherwise, the keyword 'vertical' is passed to colorbar.
    dum = strpos(strlowcase(tag_names(*self.var.internal._extra)),
'horiz')
    if max(dum, index) eq 0 then vertical =
(*self.var.internal._extra).(index) eq 0 else vertical = 1

```

```

colorbar, $
  minrange = self.var.draw.mini, maxrange = self.var.draw.maxi, $
  divisions = self.var.draw.n_lev, $
  vertical = vertical, $
  position = self.var.draw.cb_position, $
  bot=self.var.draw.bottom, ncolors=self.var.draw.ncolors - 1, $
  _extra = self -> remove_tag(*self.var.internal._extra, ['pos',
'iso'])
end

;
=====
=====
; Draw border
;
=====
=====

pro map_image::draw_border

  plots, !x.window[[0, 1, 1, 0, 0]], !y.window[[0, 0, 1, 1, 0]],
/normal
end
;
=====
=====
; Select region of interest or zoom in and out by a choosen factor (in
degree or percent)
;
=====
=====

pro map_image::zoom, _extra = _extra

  if strupcase(!d.name) eq 'PS' then begin
    print, "[map_image::zoom] ERROR: method zoom not defined for
postscript mode!"
    return
  endif
  self -> message, "[map_image::zoom]: Zoom into image"
  self -> set_var, _extra = _extra

  first_corner = 0
  second_corner = 0
  selection_done = 0

```

```

device, get_graphics = old, set_graphics = 6 ;Set xor
print, "Press left button to choose first corner."
while not selection_done do begin
    cursor, xx, yy, wait = 2, /dev
    wait, .2
    btn = !err
    if btn eq 1 then begin
        if first_corner eq 0 then begin
            if second_corner eq 1 then begin
                plots, [x0, x1, x1, x0, x0], [y0, y0, y1, y1, y0],
/device
                plots, x0, y0, /device, psym = 4
            endif
            second_corner = 0
            x0 = xx & y0 = yy
            first_corner = 1
            plots, x0, y0, /device, psym = 4
            print, "Press second button to choose second corner."
        endif else begin
            x1 = xx & y1 = yy
            plots, [x0, x1, x1, x0, x0], [y0, y0, y1, y1, y0],
/device
            second_corner = 1 & first_corner = 0
            print, "Press right button to quit or left button for new
selection."
        endelse
    endif

    if btn eq 4 and second_corner eq 1 then begin
        if x0 eq x1 or y0 eq y1 $
            then print, "Please choose rectangular area." $
            else selection_done = 1
        endif
    endwhile
device, set_graphics = old, cursor_standard=30

if x0 gt x1 then begin
    xx = x0 & x0 = x1 & x1 = xx
endif
if y0 gt y1 then begin
    yy = y0 & y0 = y1 & y1 = yy
endif

new_corners = ((convert_coord([x0, (x0+x1)/2, x1, (x0+x1)/2],
[(y0+y1)/2, y1, (y0+y1)/2, y0], /device, /to_data))[0:1, *])[*]
(*self.var.internal._extra).limit = new_corners[[1, 0, 3, 2, 5, 4,
7, 6]]
self -> project

```

```

end

;
=====
=====
; Calculate range for color bar
;
=====
=====

pro map_image::set_range

self.var.draw.mini = min(*self.var.data.image, max=maxi)
self.var.draw.maxi = maxi
if self.var.draw.mini eq self.var.draw.maxi then begin
    self.var.draw.mini = self.var.draw.mini - 1
    self.var.draw.maxi = self.var.draw.maxi + 1
    self.var.draw.n_lev = 2
endif else begin
    ;self.var.draw.maxi =
percentiles(*self.var.data.image), value = [.95])
    binsize =
10.^float(floor(aalog10((self.var.draw.maxi-self.var.draw.min i))))
    count=0
    start_here:
    count=count+1
    minout = floor(self.var.draw.mini / binsize) * binsize
    maxout = ceil(self.var.draw.maxi / binsize) * binsize
    nlev = round((maxout-minout) / binsize)
    if count gt 10 then stop
    if nlev lt 3 then begin
        binsize = binsize / 2.
        goto, start_here
    endif
    if nlev gt 8 then begin
        binsize = binsize * 2.
        goto, start_here
    endif
    self.var.draw.n_lev = nlev
    self.var.draw.mini = minout
    self.var.draw.maxi = maxout
    self -> message, string(minout, maxout, nlev, format =
'("[map_image::set_range]: Mini, maxi, n_lev: ", 3f7.2)')
endelse
end

;
=====

```

```
=====
; Set object variables
;
=====
=====
```

```
;-----
```

```
function map_image::set_limit
```

```

d1=min(*self.var.data.longitude, i1)
d2=max(*self.var.data.longitude, i2)
d3=min(*self.var.data.latitude, i3)
d4=max(*self.var.data.latitude, i4)

i1 = where(*self.var.data.longitude eq d1)
if n_elements(i1) gt 1 then i1 = i1[n_elements(i1)/2]
i2 = where(*self.var.data.longitude eq d2)
if n_elements(i2) gt 1 then i2 = i2[n_elements(i2)/2]
i3 = where(*self.var.data.latitude eq d3)
if n_elements(i3) gt 1 then i3 = i3[n_elements(i3)/2]
i4 = where(*self.var.data.latitude eq d4)
if n_elements(i4) gt 1 then i4 = i4[n_elements(i4)/2]
```

```

limit = [$
    (*self.var.data.latitude)[i1], d1, $
    d4, (*self.var.data.longitude)[i4], $
    (*self.var.data.latitude)[i2], d2, $
    d3, (*self.var.data.longitude)[i3] $
]
```

```

return, limit
end
```

```
;-----
```

```
function map_image::decomposed, sss
```

```

case (sss)[0] of
3:begin
    if (sss)[3] ne 3 then begin
        print, "[map_image::set_one_var]: For true color images,
the image must be of dimension [xsize, ysize, 3]"
        stop
    endif
    self.var.draw.true_color=1
end
2: self.var.draw.true_color = (sss)[2] eq 3
1: self.var.draw.true_color = 0
```



```

else: begin
    print, "[map_image::set_one_var]:I don't know what to do with
an image of these dimensions: "
    return, 1
end
endcase

```

```

if strupcase(!d.name) ne 'PS' then begin
    device, get_decomposed = decomp
    self.var.draw.decomposed = decomp
    self -> message, self.var.draw.true_color eq 1 $
        ? "[map_image::set_one_var]: For true color images, I will
set device, decomposed = 1" $
        : "[map_image::set_one_var]: For 8-bit images, I will set
device, decomposed = 0"
    device, decomp = self.var.draw.true_color
endif

```

```

if self.var.draw.true_color then self.var.draw.no_color_bar = 1
return, 0
end

```

```

;-----

```

```

function map_image::limit428, limit

```

```

; Transform 4-element limit into 8-element vector
;
; If no map projection is defined yet, just assume (equidistant)
; cylindrical:

```

```

x0 = limit[1] & y0 = limit[0] & x1 = limit[3] & y1 = limit[2]

```

```

if !map.projection eq 0 then begin

```

```

    new_limit = [ $
        (y0 + y1) / 2., x0, $
        y1, (x0 + x1) / 2., $
        (y0 + y1) / 2., x1, $
        y0, (x0 + x1) / 2. $
    ]

```

```

endif else begin

```

```

    ; Normal coordinates of lower left and

```

```

upper right corner:

```

```

    dum = convert_coord([x0, x1], [y0, y1], /data, /to_normal)

```

```

    ; Data coordinates of 4 points on each

```

```

side:

```

```

    x0 = dum[0, 0] & x2 = dum[0, 1] & x1 = (x2 + x0) / 2.

```

```

    y0 = dum[1, 0] & y2 = dum[1, 1] & y1 = (y2 + y0) / 2.

```

```

    dum = convert_coord([x0, x1, x2, x1], [y1, y2, y1, y0], /normal,
/to_data)

```

```

        ; Return correct limit vector
        new_limit = dum[[1, 0, 4, 3, 7, 6, 10, 9]]
    endelse
    self -> message, "[map_image::limit428]: " + $
        "Making limit a 8-element vector:"
    self -> message, "[map_image::limit428]: " + $
        "old: " + string(limit, format = '("[", 4(i4, ", "), "]')')
    self -> message, "[map_image::limit428]: " + $
        "new: " + string(new_limit, format = '("[", 8(i4, ", "), "]')')

    return, new_limit
end

;-----

pro map_image::set_one_var, tag, value, $
    no_copy = no_copy, $
    error = error

; Look for the first matching tag name in self.var.(x)
; Some variables like limit can change their shape, so they are
; stored as pointers.
; If no tag in self.var.(x) matches, the tag and value are stored in
; self.var.internal._extra

error = 0
for i = 0, n_tags(self.var) - 1 do begin
    st_names = strlowercase(tag_names(self.var.(i)))
    for j = 0, n_tags(self.var.(i)) - 1 do $
        if strpos(st_names[j], strlowercase(tag)) eq 0 then begin

            if st_names[j] eq 'image' then if self ->
decomposed(size(value)) ne 0 then begin $
                help, value
                error = 1
                return
            endif

            if size(self.var.(i).(j), /type) eq 10 $
                then if ptr_valid(self.var.(i).(j)) $
                    then *self.var.(i).(j) = value $
                else self.var.(i).(j) = ptr_new(value, no_copy = no_copy) $
                else self.var.(i).(j) = value
            if $
                strpos(st_names[j], 'ima') eq 0 and $
                self.var.draw.mini eq 0 and $
                self.var.draw.maxi eq 0 then self -> set_range

```

```

        ; Save two colours "at the bottom" for
background and void color and one "at the top" for the foreground
colour
        if st_names[j] eq 'bottom' then self.var.draw.bottom =
self.var.draw.bottom + 2
        if st_names[j] eq 'ncolors' or st_names[j] eq 'bottom' then
self.var.draw.ncolors = self.var.draw.ncolors +
(255-self.var.draw.bottom)
        return
    endif
endfor

if size(*self.var.internal._extra, /type) eq 8 then begin
    st_names = strlowercase(tag_names(*self.var.internal._extra))
    for j = 0, n_tags(*self.var.internal._extra) - 1 do begin
        if strpos(st_names[j], strlowercase(tag)) eq 0 then begin
            ;self -> message, /time,
"[map_image::set_one_var]: Setting self.var.internal._extra." +
st_names[j]
            if strpos(strlowercase(tag), 'lim') eq 0 then begin
                if n_elements(value) eq 4 then value = self ->
limit428(value)
                if value[0] eq -1 then value = self -> set_limit()
            endif
            (*self.var.internal._extra).(j) = value
            return
        endif
    endfor
    *self.var.internal._extra =
create_struct(*self.var.internal._extra, tag, value)
    ;self -> message, /time,
"[map_image::set_one_var]: Adding self.var.internal._extra." + tag;, ":
", value
endif else begin
    ;self -> message, /time,
"[map_image::set_one_var]: Creating self.var.internal._extra." + tag;,
": ", value
    *self.var.internal._extra = create_struct(tag, value)
endelse

end

;-----

function map_image::get_var, tag

; Look for the first matching tag name in self.var.(x)

```

```
; Some variables like limit can change their shape, so they are
; stored as pointers.
```

```
for i = 0, n_tags(self.var) - 1 do begin
  st_names = strlowercase(tag_names(self.var.(i)))
  for j = 0, n_tags(self.var.(i)) - 1 do begin
    if strpos(st_names[j], strlowercase(tag)) eq 0 then begin
      if size(self.var.(i).(j), /type) eq 10 then begin
        if ptr_valid(self.var.(i).(j)) then $
          return, *self.var.(i).(j) $
        else $
          return, -1
        endif else begin
          return, self.var.(i).(j)
        endelse
      endif
    endfor
  endfor
endfor
end
```

```
;-----
```

```
pro map_image::set_var, $
  error = error, $
  _extra = _extra
  if size(_extra, /type) eq 8 then begin
    names = tag_names(_extra)
    for i = 0, n_tags(_extra) - 1 do self -> set_one_var, names[i],
      _extra.(i), error = error
    endif
  end
end
```

```
;
```

```
=====
```

```
=====
```

```
; Internal subroutines
```

```
;
```

```
=====
```

```
=====
```

```
pro map_image::debug
  stop
end
```

```
;-----
```

```
function map_image::init, $
  image, lat, lon, $
```

```

        no_copy = no_copy, $
        no_project = no_project, $
        debug = debug, $
        _extra = _extra

if size(image, /type) * size(lat, /type) * size(lon, /type) eq 0 then
begin
    print, "Image or lon or lat are not defined:"
    help, image, lat, lon
    return, 0
endif
self.var.internal.debug = keyword_set(debug)
self.var.internal.time = systime(1)

if n_params() eq 3 then begin
    self -> set_one_var, 'image', image, no_copy = no_copy, error =
error
    if error eq 1 then return, 0
    self -> set_one_var, 'longitude', lon, no_copy = no_copy
    self -> set_one_var, 'latitude', lat, no_copy = no_copy
endif else no_project = 1

; First, fill all pointer variables with a default value just in case
for i = 0, n_tags(self.var) -1 do $
    for j = 0, n_tags(self.var.(i)) -1 do $
        if size(self.var.(i).(j), /type) eq 10 then $
            if not ptr_valid(self.var.(i).(j)) then begin
                ;self -> message, /time,
                "[map_image::init]: Setting " + (tag_names(self.var.(i)))[j] + " to " +
string(i*10+j, format = '(i3)')
                self.var.(i).(j) = ptr_new(i*10+j)
            endif

self -> set_one_var, 'void_index', -1
self -> set_one_var, 'void_color', 150
; Reserve colour 0 for background and colour 1 for void-color
self -> set_one_var, 'bottom', 2
self -> set_one_var, 'ncolors', 253
self.var.internal.window = !d.window ge 0 ? !d.window : 1
self.var.draw.scalef = strupcase(!d.name) eq 'PS' ? 0.03 : 1.

; Save latest color settings:
tvlct, rold, gold, bold, /get
self -> set_one_var, 'rold', rold
self -> set_one_var, 'gold', gold
self -> set_one_var, 'bold', bold
self -> set_one_var, 'color', !p.color

```

```

self -> set_one_var, 'background', !p.background
if self.var.draw.no_color_bar then begin
  !p.position = [0.1,0.1,0.93,0.9]
endif else begin
  vertical = 1
  if size(_extra, /type) eq 8 then begin
    dum = strpos(strlowcase(tag_names(_extra)), 'horiz')
    if max(dum, index) eq 0 then vertical = (_extra).(index) eq 0
else vertical = 1
endif
if vertical then begin
  ; Vertical colorbar
  !p.position = [0.24, 0.10, 0.93, 0.9]
  self.var.draw.cb_position = [0.13, 0.09, 0.16, 0.91]
endif else begin
  ; Horizontal colorbar
  !p.position = [0.07, 0.3, 0.93, 0.9]
  self.var.draw.cb_position = [0.1, 0.08, 0.9, 0.14]
endelse
endelse

```

```

self -> set_one_var, 'limit', self -> set_limit()
if not keyword_set(no_project) then $
  self -> project, _extra = _extra $
else $
  self -> set_var, _extra = _extra

```

```

return, 1
end

```

```

;-----

```

```

pro map_image::cleanup, image, lat, lon

```

```

if n_params() ge 1 then image = *self.var.data.image
if n_params() ge 2 then lat = *self.var.data.latitude
if n_params() ge 3 then lon = *self.var.data.longitude

```

```

; Find all pointers and dereference them
for i = 0, n_tags(self.var) -1 do $
  for j = 0, n_tags(self.var.(i)) -1 do $
    if size(self.var.(i).(j), /type) eq 10 then ptr_free,
self.var.(i).(j)

```

```

if strupcase(!d.name) ne 'PS' then begin

```

```

    self -> message, string(self.var.draw.decomposed, format =
'("[map_image::cleanup]: Setting device, decomposed = ", i1)')
    device, decomposed = self.var.draw.decomposed

        ; Restore original color settings:
    tvlct, self.var.draw.roid, self.var.draw.gold, self.var.draw.bold
endif
self -> message, "[map_image::colors]: !p.background: " + $
    strcompress(string(!p.background), /rem) + " --> " + $
    strcompress(string(self.var.draw.background), /rem)
self -> message, "[map_image::colors]: !p.color: " + $
    strcompress(string(!p.color), /rem) + " --> " + $
    strcompress(string(self.var.draw.color), /rem)
!p.color = self.var.draw.color
!p.background = self.var.draw.background
end

;-----

pro map_image__define

    struct = {map_image, var: {var, $
        data: {data, $
            image: ptr_new(), latitude:
ptr_new(), longitude: ptr_new() $
        }, $
        draw: {draw, $
            image_title: "", legend: "",
comment: "", $
            mini: 0., maxi: 0., n_lev:
0, $
            void_index: ptr_new(),
void_color: 0, $
            no_scale: 0, $
            no_color_bar: 0,
cb_position: [0., 0., 0., 0.], $
            magnify: 0, $
            draw_image: ptr_new(),
xoffset: 0, yoffset: 0, scalef: 1., $
            draw_image_orig: ptr_new(),
$
            decomposed: 0, true_color:
0, $
            bottom: 0b, ncolors: 0, $
            rold: bytarr(256),
gold: bytarr(256), bold: bytarr(256), $
            color: 0l, background: 0l $
        }, $
    }, $

```

```

                internal: {internal, debug: 0, window:
0, _extra: ptr_new(), time: 0d} $
                }}
end

;-----
;-----

pro map_image, image, lat, lon, _extra = _extra
    m = obj_new("map_image", image, lat, lon, _extra = _extra)
    obj_destroy, m
end

;-----
;-----

pro testdata, image, lon, lat, imgsize = imgsize

    imgsize = keyword_set(imgsize) ? imgsize : 500
    image = dist(imgsize, imgsize)
    lon = fltarr(imgsize, imgsize)
    lat = fltarr(imgsize, imgsize)
    for i = 0, imgsize-1 do lon[*, i] = findgen(imgsize) / (imgsize-1) *
40. -10.
    for i = 0, imgsize-1 do lat[i, *] = findgen(imgsize) / (imgsize-1) *
40. + 30.
end

;-----

pro check_ptr

    q = ptr_valid(count = c)
    if c gt 0 then begin
        print, c, format = '("[example1]: Still ", i3, " valid heap
variable(s)!")'
        q=ptr_valid(/cast)
        help, q, output = output
        for i = 0, n_elements(q) -1 do print, i, ": ", output
        ptr_free, ptr_valid(/cast)
    endif
end

;-----
; EXAMPLES
;-----

pro ex1, _extra = _extra

```


; Plain greyscale image

```
testdata, image, lon, lat
map_image, image, lat, lon, title = "Legend title / unit", _extra =
_extra
end
```

```
pro ex2, _extra = _extra
```

; Low resolution data (looks bad!)

```
testdata, image, lon, lat
map_image, image, lat, lon, title = "Legend title / unit", $
limit=[48, -5, 52, 0], londel=1,latdel=1, _extra = _extra
end
```

```
pro ex3, _extra = _extra
```

; Magnify image

```
testdata, image, lon, lat
map_image, image, lat, lon, title = "Legend title / unit", $
limit=[48, -5, 52, 0], londel=1,latdel=1, $
magnify = 4, $
/rainbow, mini = 220, maxi=280, /box, _extra = _extra
end
```

```
pro ex4, _extra = _extra
```

; Different projection

```
testdata, image, lon, lat
map_image, image, lat, lon, title = "Legend title / unit", $
/ortho, p0lon = 0, p0lat = 50, /iso, /rainbow, $
limit = [20, -70, 70, 180, 20, 90, -20, 10], _extra = _extra
end
```

```
pro ex5, _extra = _extra
```

; Discrete colours and different colour table

```
testdata, image, lon, lat, imgsize = 10
map_image, image, lat, lon, title = "Legend title / unit", $
magnify=20, mini=2, maxi=8, n_lev=6, discrete=[2, 3, 4, 5, 6, 7,
8], $
ctable = 15, bot = 20, ncol = 121, /box, _extra = _extra
end
```

```
pro ex6, _extra = _extra
```

```
; Discrete colors with explicitly defined colours
```

```
!quiet=1
testdata, image, lon, lat, _extra = _extra, imgs=250
map_image, lon, lat, lon, title = "Longitude / degrees", $
    red = [255, 0, 0, 255], green = [0, 255, 0, 255], blue = [0, 0,
255, 100], $
    discrete=[0, 5, 15, 20, 25], $
    /box, $
    void_index = where(lon lt 0 or lon gt 25), $
    min = 0, max=25, n_lev = 5, _extra = _extra
end
```

```
pro ex7, _extra = _extra
```

```
; Discrete colors, check for exact discrimination
```

```
testdata, image, lon, lat
map_image, lon, lat, lon, title="Longitude / degrees", $
    mini = 4, maxi = 6, n_lev = 4, limit = [51, 4, 53, 6], londel =
.5, latdel = .5, magnif = 2, $
    format = '(f3.1)', /box, $
    red = [122, 0, 255, 70], green = [0, 255, 70, 122], blue = [255,
70, 122, 0], $
    discrete=[4, 4.5, 5, 5.5, 6.], _extra = _extra

end
```

```
pro ex8, _extra = _extra
```

```
; Zoom into image
```

```
testdata, image, lon, lat
m = obj_new("map_image", image, lat, lon, /no_copy, title = "Legend
title / unit", /rainbow, _extra = _extra)
m -> zoom, win = 2, magnify = 2
obj_destroy, m
check_ptr
!quiet=0
end
```

```
pro ex9, _extra = _extra
```

```
; Multiple "overpasses"
```

```

!quiet=1
testdata, image, lon, lat
m = obj_new($
    "map_image", image, lat, lon, $
    title = "Legend title / unit", $
    limit = [0, -65, 80, 10, 0, 35, -30, 10], $
    /no_draw, $
    /rainbow, $
    _extra = _extra)
m -> project, image = image, lon = lon, lat = lat - 50, /no_erase,
/no_draw
m -> project, image = image, lon = lon - 50, lat = lat - 50,
/no_erase, /no_draw
m -> project, image = image, lon = lon - 50, lat = lat, /no_erase,
/no_draw
m -> display
obj_destroy, m
check_ptr
!quiet=0

end

```

```

pro ex10, _extra = _extra

```

```

; Multiple images

```

```

!quiet=1
testdata, image, lon, lat
map_image, image, lat, lon, pos = [.05, .22, 0.45, .50], londel = 10,
latdel = 10, /no_col, /rainbow, _extra = _extra
map_image, image, lat, lon, /noerase, pos = [.05, .22, 0.45, .50] +
[0.5, 0, 0.5, 0], londel = 20, latdel = 20, /sinu, /no_col, /rainbow,
_extra = _extra
map_image, image, lat, lon, /noerase, pos = [.05, .22, 0.45, .50] +
[0, 0.45, 0, 0.45], /ortho, p0lon = 0, p0lat = 50, /iso, /rainbow, $
    limit = [20, -70, 70, 180, 20, 90, -20, 10], /no_col, _extra =
_extra
m = obj_new("map_image", image, lat, lon, /noerase, pos = [.05, .22,
0.45, .50] + [0.5, 0.45, 0.5, 0.45], londel = 2, latdel = 2, limit =
[55, -10, 60, -5, 55, 0, 50, -5], /no_col, magnif = 2, /rainbow, _extra
= _extra)
m -> colorbar, cb_position = [.04, .07, .96, .1], /horizontal, title
= 'Legend title / unit'
obj_destroy, m
check_ptr
!quiet=0
end

```

```

pro ex11, _extra = _extra

; True color image

testdata, image, lon, lat
map_image, [[[bytsc1(lon)]]], [[[bytsc1(lat)]]], [[[bytsc1(image)*0b]]],
lat, lon, $
; limit = [28, -15, 72, 35], $
_extra = _extra

end

pro ex12

; Logarithmic colour scale

testdata, image, lon, lat
image = image / max(image)
image = exp(image)
image = (image-min(image)) / max(image) * 20 + .1

map_image, win = 1, image, lat, lon, /rainbow, min = 0, max = 20,
n_lev = 4

map_image, win = 2, alog10(image), lat, lon, /rainbow, $
mini = alog10(0.1), maxi = alog10(100), $
/ylog, range = [0.1, 100], $
format = '(f5.1)', div = 3, minor = 10
end

;-----

pro map_image_demo, _extra = _extra

s = ""
print, "[Example 1]: Just a plain greyscale image." & ex1, _extra =
_extra & read, s, prompt = "    <ENTER> "
;write_png, "ex01.png", tvrd(true=1)
print, "[Example 2]: Now, zoom into the image with the LIMIT
keyword." & ex2, _extra = _extra
;write_png, "ex02.png", tvrd(true=1)
print, "[Example 2]: This looks comparably bad, so let's try MAGNIFY"
& read, s, prompt = "    <ENTER> "
print, "[Example 3]: Much better. We also chose a rainbow colour bar"
print, "[Example 3]: and new min and max values. Plus the box around
with the"

```

```

    print, "[Example 3]: lat / lon values." & ex3, _extra = _extra &
read, s, prompt = "    <ENTER> "
;write_png, "ex03.png", tvrd(true=1)
    print, "[Example 4]: Now with a different projection:" & ex4, _extra
= _extra & read, s, prompt = "    <ENTER> "
;write_png, "ex04.png", tvrd(true=1)
    print, "[Example 5]: A different colour table and discrete values:" &
ex5, _extra = _extra & read, s, prompt = "    <ENTER> "
;write_png, "ex05.png", tvrd(true=1)
    print, "[Example 6]: Again discrete colours, but this time with
explicitely defined colours:" & ex6, _extra = _extra & read, s, prompt
= "    <ENTER> "
;write_png, "ex06.png", tvrd(true=1)
    print, "[Example 7]: And again, now check for exact discrimination of
colours / values:" & ex7, _extra = _extra & read, s, prompt = "
<ENTER> "
;write_png, "ex07.png", tvrd(true=1)
;print, "[Example 8]: Interactively zoom into an image:" & ex8, _extra
= _extra & read, s, prompt = "    <ENTER> "
    print, "[Example 9]: Plot multiple datasets in one projection:" &
ex9, _extra = _extra & read, s, prompt = "    <ENTER> "
;write_png, "ex09.png", tvrd(true=1)
    print, "[Example 10]: Plot multiple datasets in different
projections, but with a common, horizontal colour bar:" & ex10, _extra
= _extra & read, s, prompt = "    <ENTER> "
;write_png, "ex10.png", tvrd(true=1)
    print, "[Example 11]: And finally, a true colour image:" & ex11,
_extra = _extra
;write_png, "ex11.png", tvrd(true=1)
    print, "[Example 12]: Use logarithmic scaling:" & ex12, _extra =
_extra
;write_png, "ex11.png", tvrd(true=1)
end

```

```

#####
# win.pro
#####

pro win , i,xsize=xsize,ysize=ysize,_extra=_extra
;-----
;+
; NAME:
;   win
; PURPOSE:
;   win is a combination of wset and window
; if the window i already exists wset is used
; otherwise the window will be created.

```

```

; if device is postscript, nothing happens
;
;
; CATEGORY:
; Graphics
;
; CALLING SEQUENCE:
; win,1
;
; KEYWORDS:
; all window/wset keywords
;
; MODIFICATION HISTORY:
;   Written by R. Preusker, Jan, 2001.
;
; Copyright (C) 2001, Freie Universitaet Berlin
; This software may be used, copied, or redistributed as long
; as it is not sold and this copyright notice is reproduced on
; each copy made. This routine is provided as is without any
; express or implied warranties whatsoever.
;
; rene@amor.met.fu-berlin.de
;-

if STRLOWCASE(!d.name) eq "ps" then return
if not (keyword_set (xsize)) then xsize=400
if not (keyword_set (ysize)) then ysize=300

if (i eq 0) and (!d.window eq -1) then
window,i,xsize=xsize,ysize=ysize,_extra=_extra

catch, errorstat

if errorstat ne 0 then window,i,xsize=xsize,ysize=ysize,_extra=_extra
else wset,i

end

```
