IDL read routine for DOMINO HDF-EOS5 data file

Ruud Dirksen and Folkert Boersma, KNMI, 23 april 2008

The file read_L2_no2.pro contains an IDL function which reads a DOMINO hdfeos5 file and stores the contents of the data file in a data structure called data. The function read_L2_no2 uses the IDL native <h5_parse> function and reorganizes the multi-nested structure which is returned by that function into a more surveyable data structure. The purpose of this document is to offer a brief description of how to use the function and the resulting data structure. The code has been tested in IDL version 6.4.1. Entries such as the tropospheric NO2 column are stored as data.vcdtrop, etc.

To compile the function type

```
IDL>.com read_L2_no2
```

To read a DOMINO hdfeos5 data file type

```
IDL>data=read_L2_no2(filename)
```

filename is a string containing the path and name of the file to be read in. So for example:

```
IDL>data=read_L2_no2('/nobackup/users/boersma/data/OMI-Aura_L2-
OMDOMINO_2006m0129t2141-o08208_v003-2008m0407t010239.he5')
```

(in this case don't forget the parentheses when passing filename!)

This stores the contents of the HDF-EOS5 in a structure called data. The contents of this structure can be checked by typing:

```
IDL>help,data
DATA   STRUCT      = -> <Anonymous> Array[60,1644]
```

i.e. it tells us that our datastructure has the format of the HDF-EOS5 swath; 60 across-track pixels times 1644 along-track pixels.

By typing

```
IDL>help,data,/str
```

We get to see the contents of the structure. Most data fields abbreviations are self explanatory. Some examples:

```
IDL> data(34,657).cp_ll
```
gives the lat and lon for the lower left corner of the pixel.

```
IDL> print,*data(4,367).kernel
```
gives 35-layered averaging kernel for this pixel. Because the kernel datafield is of type pointer you should dereference this field by typing an asterix (*) before the parameter name. The reason for using a pointer for this data field is memory usage. The kernel constitutes an array of 34/35 elements per pixel, which can grow to large proportions when handling a large dataset. If you are not interested in using the averaging kernel, we recommend commenting out line 36 (kernel=). By doing so, no memory for the averaging kernel is allocated.

Be aware of memory leaks when handling multiple orbit files, if you discard an orbit make sure to free the memory that was allocated for the averaging kernel. This can simply be done by the command

```
IDL>ptr_free, data.kernel
```

Failure to properly free allocated memory can cause memory overload which will result in lower performance and eventually in a halting of your IDL program.