RADS version 4.2.1

User Manual

Remko Scharroo

18 April 2016

This document was typeset with $\LaTeX 2_\varepsilon$.
The layout was designed by Remko Scharroo © 1993–2015

Contents

Chapter 1
Introduction

This document describes the layout and use of the Radar Altimeter Database System (RADS), Version 4. RADS was first developed at Delft University of Technology's Department of Aerospace Engineering, and remains a joint development with NOAA Laboratory for Satellite Altimetry and EUMETSAT.

The Radar Altimeter Database System is composed of three elements:

- A few hundred gigabytes of altimeter data files from missions stretching from Geosat to whatever altimeter data was made available in the last few days;
- A set of software tools (object library and executables);
- Configuration files.

So apart from the actual altimeter data, RADS provides a suite of applications and subroutines that simplify the reading, editing, handling and analysing of data from numerous radar altimeters. Although the actual content and layout of the underlying data products do not have to be identical for all altimeters, the user interface is. Also, the data base is easily expandable with additional data and/or additional corrections without impact to the user interface, or to the software in general. In fact, only in very few cases the core software will need to be adjusted and recompiled, in even fewer cases adjustments to the actual tools will be required. Most changes can be covered by changes in the configuration file.

The data base consists of netCDF files, one for each satellite pass (half a revolution starting and ending close to the poles). Ascending passes have odd numbers, descending passes even numbers. The pass numbering increases consecutively within a repeat cycle.

In case of exact repeat missions the satellite returns to the same ground track every repeat cycle. For Jason-2, for example, this is after 254 passes, when the pass number starts over at 1. Which pass is number 1 is based on the longitude of the equator crossing (ascending node). Thus all passes with the same pass number are collinear.

For non-repeat missions or those with very long repeat cycles (like CryoSat-2 or the Jason-1 Extended Mission), we created "sub-cycles" of a manageable length. There too passes with the same pass number are nearly collinear. Note that the length of the "sub-cycle" may change for cycle to cycle in a kind of dance-step manner.

Each netCDF data file contains the actual (binary) data as well as the meta data that describe the contents (data type, units, creation history, etc.) The naming convention for the files is `SSpPPPPcCCC.nc`, where `SS` is an abbreviation for the satellite (altimeter), `PPPP` is the pass number, `ccc` is the cycle number, and `nc` is the extension, a standard convention for netCDF data files.

The data files are grouped in one directory for each cycle, named `cccc`. These cycle directories are then grouped into one directory for each mission phase, which are finally part of one directory per satellite. For example, the data file for pass 801 of cycle 150 in ERS-1's tandem mission is `$RADSDATAROOT/e1/g/c150/e1p0801c150.nc`, where `$RADSDATAROOT` is the root directory of the RADS data base.

To read and manipulate the data, you can use standard netCDF tools, like `ncdump` (that comes with the netCDF package), GMT (Generic Mapping Tools), nco (NetCDF Operators). But more suitable is the use of the RADS subroutine library and programs. The library is the basis for all data utilities provided with RADS and can also be used to create other programs to the user's convenience. For a description on each of the subroutines in the library and on how to create your own program see Appendix 6. In addition, a number of handy utilities are provided to do some of the most essential jobs (Chapter 5).

Whether you are using the routines, or the provided utilities, you will have to know how the data handling system of RADS works. It is not essential that you understand the intrinsics of the data files, but it is highly recommended that you familiarise yourself with the way the data can be manipulated, selected and edited *on the fly* by the RADS routines. Basically, the RADS routines can take you a lot of work out of your hands, provided you have read Chapter 4.

Before going into the details of RADS, the software and the data have to be installed on your computer. Chapter 2 guides you through the process of software installation, and Chapter 3 tells you how to keep the database up to date.

Chapter 2

RADS software installation

In order to work efficiently with the RADS data base you are required to install the software (subroutine library, utilities, scripts, and configuration files). This we will tackle in this Chapter. Once you are done with that at least part of the data base needs to be copied onto your hard disk (or another mounted device), which will be described in the next Chapter.

2.1 Prerequisites

In order to install and run the RADS software you need a few things installed on your system:

- A unix platform (for example Linux or Mac OS X).
- The make command.
- A Fortran 90 compiler. RADS is known to compile with gfortran, f90, f95, xlf90, xlf95, ifort.
- The netCDF library (version 4) and module file compiled with the Fortran 90 interface. Of course, netCDF comes with its own dependencies (like HDF5 and szip). Please figure out where to find the netCDF module file netcdf.mod and the netCDF C library libnetcdf and Fortran library libnetcdff before you continue.
- Optionally, the git program.
- For downloading and synchronising the data base, the rsync program.

2.2 Download the source code

The source code can be downloaded as a bundle (zip or tarball) from GitHub or can be synchronised directly with the github server with the git program. The two methods are described below in Sections 2.2.1 and 2.2.2.

You can put the source code anywhere you like. We will later configure where things will be installed. After downloading the software, continue with the configuration, compilation, and installation steps in Sections 2.3 through 2.5.

It is recommended to regularly check for updates of the RADS source code and recompile if necessary.

2.2.1 Download the bundle from GitHub

To download the latest bundle of the source code, simple go to https://github.com/remkos/rads/releases/latest. There you will find the latest release notes, and links for the downloading of the bundle, either as a zip file, or as a compressed tarball.

You can extract the software in place, or anywhere you want by running:

```
$ tar -xvzf rads-v4.2.1.tar.gz
```

or

```
$ unzip rads-v4.2.1.zip
```

This will create a directory called `rads-v4.2.1`.

2.2.2 Software synchronisation with git

Git is a version control system that helps to administrate software development projects on distributed systems (or at least by distributed users), avoiding problems of accidentally wiping out each others changes. Also, it is a very practical tool for distributing trees of software to others, who then can make their own changes without running the risk of accidentally overwriting them when a new update is provided. Git can merge those changes, and alerts you of that happening.

You need to have at least the executable git installed on your system to connect to the GitHub repository. This program comes installed by default on Mac OS X and most Linux and Unix systems.

First you need to 'clone' the code from the GitHub server onto your machine:

```
$ git clone https://github.com/remkos/rads
```

This downloads all the code and puts it into a directory called `rads`. This needs to be done only once.

Later on you can bring the source on your machine up to date by going into the `rads` directory and executing:

```
$ git pull -t origin master
```

although it is much simpler to just use:

```
$ make update
```

2.3 Software configuration

Now we are going to determine where the software executables, library, and data is going to be stored. For this we run the configure in the source directory (`rads-v4.2.1` if you downloaded the tarball, or `rads` if you used git). The program configure will allow you to specify where you want things installed and also determines which Fortran compiler you have and what special options are needed for your platform.

By default, configure will install everything under the directory were it resides itself. It will create directories:

bin for the executables (both binaries and scripts)

include for the Fortran 90 module files to be used with the RADS library

lib for the RADS library

share for the system independent data: the satellite data and configuration files. This one particularly, you might want to put somewhere else, on a dedicated disk, for example.

Normally, you would need to tell configure only where you want to install the aforementioned directories. The rest, like where to find your Fortran compiler and the netCDF library, are things that configure should be able to figure out by itself, using the nf-config script, for example. Therefore, you will only have to specify the root directory for the installation (prefix) and likely the place where you want the RADS altimeter data to reside or where they are already residing (datadir). Run, for example:

```
$ configure --prefix=/usr/local --datadir=/rads/data
```

The first argument to configure specifies that the bin, include and lib directory are to be put under /usr/local. The second argument specifies the directory for the data and configuration file (which could be on a server for more systems to use). Still a directory share is created under /usr/local to contain the manuals.

If, for whatever reason, configure cannot find a Fortran compiler or the netCDF libraries on its own, you need to specify the location of the Fortran compiler and the netCDF library and include files. Here is an example:

```
$ configure FC=/sw/bin/gfortran \
  --with-netcdf-inc=/sw/lib/netcdf-gfortran/include \
  --with-netcdf-lib=/sw/lib:/sw/lib/netcdf-gfortran/lib \
  --prefix=/usr/local --datadir=/rads/data
```

The first argument to configure specifies the location of the fortran compiler, while the second identifies the directory where we can find netcdf.mod. The third argument specifies the two directories that contain the netCDF C library (libnetcdf) and netCDF Fortran library (libnetcdff), separated by a colon. If these two are merged, or in one directory, you can just use one directory name.

The configure program also tests if your Fortran compiler is ready for Fortran 90 and can compile with the netCDF library. If you have problems, you may need to review the options you gave to configure, and make sure that configure picked the same compiler that was used to compile the netCDF library. Finding the nf-config command on your system may be pivotal.

Run `configure --help` to get more info.

2.4 Software compilation

Now that your system is configured, it should be easy to compile the software. Just run in the source directory (where you also ran configure):

```
$ make
```

It will compile and link the programs in the subdirectory src, but not those in devel. The latter are only provided to you to get a feel of how the RADS altimeter database was created. You will not be able to compile or link those programs, as essential routines have been left out.

If you have problems compiling, you may need to tweak one of the makefiles, config.mk. Please let us know about it, so that we can change the configure program accordingly. You can do this at the issue tracker on the RADS GitHub page: https://github.com/remkos/rads/issues.

2.5 Installation

To install the software, configuration file, and manuals in the places discussed in Section 2.3, run the following command in the directory where configure resides:

```
$ make install
```

Now you can continue with the mirroring of the data files.

Chapter 3

RADS data mirroring

RADS now exceeds 400 GBytes of data. It virtually impossible to copy all of it in one go, or copy all of it every time that updates have been made. To facilitate the updating, it is recommended to use the rsync program. This program will determine by it self which files are updated and will update only those. In fact, it will transfer only those parts of the files that are actually changed. This provides a significant speed benefit when, for example, an extra data field is added.

You need to have at least the executable rsync installed on your system to use rsync. In case of Linux machines, simply install the rsync package available on most distributions. The program rsync comes standard with Mac OS X, or can be obtained from http://rsync.samba.org.

The rsync command will download the data from the rsync server at the Delft University of Technology in The Netherlands. This server is setup such that it will allow you to access only the RADS data and software. It will not allow you to log in to the server as a common user. Thus, setting up ssh key pairs is not possible.

Let us start, for example, to synchronise the Jason-2 data. The subdirectory for the Jason-2 data is `j2` (See Table 3.1 for all 2-character abbreviations of the altimeter missions). To get all the Jason-2 data, you will type the following commands (still assuming you have your data in /rads/data):

```
$ cd /rads/data
$ rsync -avz --del radsuser@rads.tudelft.nl::rads/data/j2 .
```

At the beginning rsync will ask you to enter the password for `radsuser`. It will have been provided to you when you registered as a user.

Apart from the satellite specific directories, there is a directory that contains configuration files, that help RADS to read the data files. These files are also installed in the same place, when installing the software, but you can download them from the rsync server as well. Be sure to keep these feels up to date.

```
$ cd /rads/data
$ rsync -avz --del radsuser@rads.tudelft.nl::rads/data/conf .
```

If you are patient, and want to get all of the data at once, you can perform the following commands:

```
$ cd /rads
$ rsync -avz --del radsuser@rads.tudelft.nl::rads/data .
```

Altimeter	Abbr.	Nr	Alternatives	References
GEOS 3	g3	1	ge3 geos-3 geos3	(not included in RADS)
Seasat	ss	2	sea seasat-a	(not included in RADS)
Geosat	gs	3	geo geosat	
ERS-1	e1	4	er1 ers-1 ers1	[*Francis*, 1990; *Francis et al.*, 1991]
TOPEX	tx	5	top topex	[*Fu et al.*, 1994]
Poseidon	pn	6	pos poseidon	
ERS-2	e2	7	er2 ers-2 ers2	[*Francis et al.*, 1995]
GFO	g1	8	gfo gfo-1 gfo1	
Jason-1	j1	9	ja1 jason-1 jason1	[*Ménard et al.*, 2003]
Envisat	n1	10	en1 envisat	
Jason-2	j2	11	ja2 jason-2 jason2	[*Lambin et al.*, 2010]
CryoSat-2	c2	12	cs2 cryosat-2 cryosat2	[*Wingham et al.*, 2006]
SARAL	sa	13	sa srl saral altika	
Jason-3	j3	14	ja3 jason-3 jason3	(limited access in RADS)
HY-2A	2a	15	h2a hy-2a hy2a	(not included in RADS)
Sentinel-3A	3a	16	s3a sentinel-3a sentinel3a sntnl-3a	(in RADS summer 2016)
Sentinel-3B	3b	17	s3b sentinel-3b sentinel3b sntnl-3b	(to be launched end 2017)

Table 3.1 Abbreviation and numbers used for the various altimeter missions.

If, for whatever reason, the mirroring is interrupted, you can simply start it again, and it will continue where it left off. If you have a recent version of the rsync program, we recommend that you use the option `--del` instead of `--delete`, as it speeds up the process significantly.

There are one more directory that may be of interest, but is not essential. The `/rads/tables` directory contains a number of lists: lists of the time intervals of passes and cycles, and lists of the data available for each satellite.

To mirror this directories use rsync:

```
$ cd /rads
$ rsync -avz --del radsuser@rads.tudelft.nl::rads/tables .
```

If you need to use rsync regularly to synchronise the RADS data base and you do not want to enter the password every time, you can set up the environment variable `RSYNC_PASSWD` by one of the following methods (depending on the shell):

```
   export RSYNC_PASSWD=radspasswd    # under sh or bash
   setenv RSYNC_PASSWD radspasswd    # under csh or tcsh
```

If you are using the bash or sh shell, you can do it all in one line, for example:

```
   RSYNC_PASSWD=radspasswd \
      rsync -avz --del radsuser@rads.tudelft.nl::rads/data .
```

Obviously, the password is **not** simply `radspasswd`.

Chapter 4
RADS data management

This Chapter describes the basic functionalities of the data management system of RADS. These functionalities are part of the RADS utilities as well as the RADS subroutine library on which the utilities are based.

4.1 Preparations before using the data base

The use of RADS starts with the definition of the environment variable `RADSDATAROOT`, such that it points to the root of the RADS data base. For example (for bash users):

```
$ export RADSDATAROOT=/rads/data
```

If you already specified this directory in the configure step, then you do not have to set the `RADSDATAROOT` directory.

It is also practical to include /usr/local/bin (or where ever you installed the binaries) in your executable search path, so that existing executables can be used. However, this is not essential, just practical.

4.2 Common functionalities

A subroutine library is created to facilitate the data reading, conversion to SI units, editing and the construction of sea level anomalies. Based on these routines, several programs (utilities) are created to list or manipulate the contents of the data base. Since these programs share the same routines, much of their functionalities are the same, as well as their user interface. In addition, a number of developer tools are available to create the data base. These programs will be of less concern to the users, and are not yet explained in this manual. Their code is only provided for reference, and are not intended to be compiled on your system.

Common to all RADS utilities is the internal data selection, editing, and the ability to construct the sea level anomaly (or any other fields) *on the fly*. The construction of the sea level anomaly includes a number of arithmetic expressions (adding and subtracting), plus applying a number of selection criteria that can flag the sea level anomaly as invalid. Both the ability to do arithmetic (even beyond mere adding and subtracting) and editing are built into the software library, so that they can be shared by the various RADS utilities.

Even when writing their own program, the user does not have to generate code to construct a sea level anomaly or to do editing. At the same time that the sea level anomaly is constructed, the data is edited based on system-wide, user, or local preferences. These preferences can be specified at several levels, either in XML configuration files, or by command line options. The order in which these preferences are processed is the following:

- System-wide general preferences, found in `$RADSDATAROOT/conf/rads.xml`.
- General user preferences, found in a file `~/.rads/rads.xml`, if available.
- General local preferences, found in a file `rads.xml` in the current working directory.
- Preference files indicated by the `-X` or `--xml` option on the command line.
- Other command line arguments, such as `-L` or `--limits`.

So the command line arguments overrule the local preferences and which overrule the user preferences and eventually the system-wide preferences.

The XML configuration file is "human readable" and is described in the next section.

4.3 RADS configuration file

The RADS configuration file rads.xml controls:

- Information about the various altimeter missions, such as repeat period, number of passes per cycle, etc.
- Information about all the variables available for each mission, including editing criteria, format for writing to ascii, and the complete information needed to store the variable upon creation of the database.
- Optionally, description of variables that can be created *on the fly* from other variables.

The configuration file is a structured XML file, comprised of a number items and blocks, of the following shape:

```
<block>
    <item option="value">....</item>
    <tag>....</tag>
</block>
```

As usual in an XML file, `<tag>` starts an item with name `tag` and `</tag>` ends it. Any content in between is the value, or are the values associated with this tag. Generally, the line can be broken by carriage returns and leading spaces are ignored.

4.3.1 Tags in the configuration file

The various names of the tags used in the configuration file are described below.

if is followed by a condition such as `sat="j1 j2"`, which means that the contents of the block following this `<if>` applies only to the missions `j1` and `j2`. The selection can be negated by starting it with an exclamation point, i.e. `sat="!j1 j2"` means that the following block applies to all missions *except* `j1` *and* `j2`.

elseif, else can follow an if-block. For example:

```
<if sat="j1 j2">
    ... for j1 and j2 ...
```

```
        </if>
        <elseif sat="e1 e2">
            ... for e1 and e2 ...
        </elseif>
        <else>
            ... all other missions ...
        </else>
```

global_attributes specifies the global attributes that are always to be written to the output file, line by line. The first word on each line is the name of the global attribute, the rest is its value.

satellites specifies, line by line, the 2- and 3-character abbreviations for the altimeter missions. The rest of each line is used to specify alternative names which can be searched for substrings.

var identifies a block of tags that describe a single variable. The tag needs to include an option such as `name="alt_gdre"` to specify the variable name, and optionally (for example) `field=425` which specify the field numbers used in RADS3. A condition like `sat="j1 j2"` can also be added as an option in this tag.

alias specifies an alias (i.e. an alternative name) for a previously specified variable. The tag should at least include an option such as `name="alt"` to specify the name of the alias, and optionally the field number used in RADS3 (e.g. `field=4`). Again, a condition like `sat="j1"` can be added. If more than one value is given, then it means that the alias will point to the first variable, unless it is not available, then it points to the second. For example:

```
<alias name="alt" field="4" sat="j2">alt_gdre alt_gdrd</alias>
```

means that if one uses the variable `alt` for Jason-2, then the variable `alt_gdre` is processed. If determining this variable was unsuccessful, then `alt_gdrd` is used instead.

4.3.2 Tags within a var-block

The following tags can only occur within a var-block (with one exception, see below):

long_name specifies the description of the variable, as well as the `long_name` attribute to be written to the netCDF data files.

standard_name, source, comment specify the `standard_name`, `source`, and `comment` attributes to be written to the netCDF data files.

units specifies the units of the variable as used in any output, as well as the `units` attribute in the netCDF data files.

flag_values specifies the meanings of a flag, counting up from 0. For example `yes no` means that 0 is the be interpreted as ”yes”, 1 as ”no”.

flag_mask specifies the meanings of a flag word made up of bits. For example `left front up` means that when the LSB is raised ”left” is true, and the next bits indicate if ”front” and ”up” are true.

limits specifies the range of ”good” values. Any value less than the lower limit or greater than the upper limit result in the variable to be set to NaN. In case the variable is a flag_mask, then the first value indicates the mask of bits that should not be set, the second value the bits that should be set, otherwise the result will yield NaN.

plot_range specifies a suggested range for plotting the variable. It is not used for any processing, except for the radsvar program.

parameters is used for the RADS4 database creation and has no impact on any user programs.

data specifies the variable name as used in the input RADS netCDF data files (normally the same as the variable name), but could also be used to specify a mathematical statement that derives the variable from others, or to interpolate values from a grid. More about this in Section 4.3.3.

quality_flag indicates which variables are checked to determine the quality of the current variable. If any of the specified variables lead to NaN (not-a-number), the corresponding value of the current variable is set to NaN as well.

dimension is the dimension of the variable. Default is 1-dimensional. Some variables can be 2-dimensional.

format is the Fortran format specification for ASCII output of the variable.

compress specifies the data type (i.e., `int1`, `int2`, `int4`, `real`, or `dble`) and optional `scale_factor` and `add_offset` to be used in the output netCDF files. This applied only the output files. Input netCDF files may be different.

Because you may want to overrule just one line in a var-block, one can stick the option `var="varname"` in the above tags. For example,

```
<var name="alt_gdre">
    <compress sat="j1 j2 j3 tx">int4 1e-4 1300e3</compress>
</var>
```

is equivalent to

```
<compress var="alt_gdre" sat="j1 j2 j3 tx">int4 1e-4 1300e3</compress>
```

4.3.3 The data-tag

The data-tag can be used in a variety of ways.

- The simplest form, explained above, is that the value indicates the name of the netCDF variable in the RADS data files. For example: `<data>alt_gdre</data>`.
- If more than one value is given (i.e. it contains at least one space), then it will be recognised as a mathematical statement, in ”Reverse Polish” notation. A number of mathematical operators are available, as described in Table 4.1. This can be very practical to determine on the fly the difference between two variables, or a combination of many (such as the sea level anomaly). For example `<data>wet_tropo_rad wet_tropo_ecmwf SUB</data>` computes the difference between the radiometer and ECWMF wet tropospheric corrections.
- branch *To be completed*
- grid, grid_s, grid_q *To be completed*
- constant *To be completed*

4.3.4 Tags for mission definitions

To be completed.

Operator	Description
x y SUB a	a = x - y
x y ADD a	a = x + y
x y MUL a	a = x * y
PI a	a = pi
E a	a = exp(1)
x POP	remove last item from stack
x NEG a	a = −x
x ABS a	a = |x|
x INV a	a = 1/x
x SQRT a	a = sqrt(x)
x SQR a	a = x*x
x EXP a	a = exp(x)
x LOG a	a = ln(x)
x LOG10 a	a = log10(x)
x SIN a	a = sin(x)
x COS a	a = cos(x)
x TAN a	a = tan(x)
x SIND a	a = sin(x) [x in degrees]
x COSD a	a = cos(x) [x in degrees]
x TAND a	a = tan(x) [x in degrees]
x SINH a	a = sinh(x)
x COSH a	a = cosh(x)
x TANH a	a = tanh(x)
x ASIN a	a = arcsin(x)
x ACOS a	a = arccos(x)
x ATAN a	a = arctan(x)
x ASIND a	a = arcsin(x) [a in degrees]
x ACOSD a	a = arccos(x) [a in degrees]
x ATAND a	a = arctan(x) [a in degrees]
x ASINH a	a = arcsinh(x)
x ACOSH a	a = arccosh(x)
x ATANH a	a = arctanh(x)
x ISNAN a	a = 1 if x is NaN; a = 0 otherwise
x ISAN a	a = 0 if x is NaN; a = 1 otherwise
x RINT a	a is nearest integer to x
x NINT a	a is nearest integer to x
x CEIL a	a is nearest integer greater or equal to x
x CEILING a	a is nearest integer greater or equal to x
x FLOOR a	a is nearest integer less or equal to x
x D2R a	convert x from degrees to radian
x R2D a	convert x from radian to degrees
x YMDHMS a	convert seconds of 1985 to format YYYYMMDDHHMMSS
x SUM a	a(i) = x(1) + ... + x(i) while skipping all NaN
x DIF a	a(i) = x(i)-x(i-1); a(1) = NaN
x DUP a b	duplicate the last item on the stack
x y DIV a	a = x / y
x y POW a	a = x^y
x y FMOD a	a = x modulo y
x y MIN a	a = the lesser of x and y
x y MAX a	a = the greater of x and y
x y ATAN2 a	a = arctan(x) taking into account the quadrant as determined by y
x y HYPOT a	a = sqrt(x*x+y*y)
x y R2 a	a = x*x + y*y
x y EQ a	a = 1 if x == y; a = 0 otherwise
x y NE a	a = 0 if x == y; a = 1 otherwise
x y LT a	a = 1 if x < y; a = 0 otherwise
x y LE a	a = 1 if x ≤ y; a = 0 otherwise
x y GT a	a = 1 if x > y; a = 0 otherwise
x y GE a	a = 1 if x ≥ y; a = 0 otherwise
x y NAN a	a = NaN if x == y; a = x otherwise
x y AND a	a = y if x is NaN; a = x otherwise
x y OR a	a = NaN if y is NaN; a = x otherwise
x y IAND a	a = bitwise AND of x and y
x y IOR a	a = bitwise OR of x and y
x y BTEST a	a = 1 if bit y of x is set; a = 0 otherwise
x y AVG a	a = 0.5*(x+y) [when x or y is NaN a returns the other value]
x y DXDY a	a(i) = (x(i+1)-x(i-1))/(y(i+1)-y(i-1)); a(1) = a(n) = NaN
x y EXCH a b	exchange the last two items on the stack (NaNs have no influence)
x y z INRANGE a	a = 1 if x is between y and z; a = 0 otherwise (also in case of any NaN)
x y z BOXCAR a	a = filter x along monotonic dimension y with boxcar of length z (NaNs are skipped)
x y z GAUSS a	a = filter x along monotonic dimension y with Gauss function with sigma z (NaNs are skipped)

Table 4.1 Math operators that can be used in the data-tag. Left of the operator (in uppercase) are the input value(s); on the right of the operator the output value(s).

Chapter *5*

RADS utilities

Chapter *6*

RADS library

6.1 Module rads

6.1.1 rads

SUMMARY:

```
 RADS main module
```

SYNOPSIS:

```
module rads
use typesizes
use rads_grid, only: grid

! * Parameters
! Dimensions
integer(fourbyteint), parameter :: rads_var_chunk = 100, rads_varl = 40, &
    rads_naml = 160, rads_cmdl = 320, rads_strl = 1600, rads_hstl = 3200, &
    rads_cyclistl = 50, rads_optl = 50, rads_max_branches = 5
! RADS4 data types
integer(fourbyteint), parameter :: rads_type_other = 0, rads_type_sla = 1, &
    rads_type_flagmasks = 2, rads_type_flagvalues = 3, rads_type_time = 11, &
    rads_type_lat = 12, rads_type_lon = 13, rads_type_dim = 14
! RADS4 data sources
integer(fourbyteint), parameter :: rads_src_none = 0, rads_src_nc_var = 10, &
    rads_src_nc_att = 11, rads_src_math = 20, rads_src_grid_lininter = 30, &
    rads_src_grid_splinter = 31, rads_src_grid_query = 32, &
    rads_src_constant = 40, rads_src_flags = 50, rads_src_tpj = 60
! RADS4 warnings
integer(fourbyteint), parameter :: rads_warn_nc_file = -3
! RADS4 errors
integer(fourbyteint), parameter :: rads_noerr = 0, &
    rads_err_nc_file = 1, rads_err_nc_parse = 2, rads_err_nc_close = 3, rads_err_memory = 4, &
    rads_err_var = 5, rads_err_source = 6, rads_err_nc_var = 7, rads_err_nc_get = 8, &
    rads_err_xml_parse = 9, rads_err_xml_file = 10, rads_err_alias = 11, rads_err_math = 12, &
    rads_err_cycle = 13, rads_err_nc_create = 14, rads_err_nc_put = 15
! Additional RADS4 helpers
character(len=1), parameter :: rads_linefeed = char(10), rads_noedit = '_'
! RADS3 errors or incompatibilities
integer(fourbyteint), parameter :: rads_err_incompat = 101, rads_err_noinit = 102
integer(twobyteint), parameter :: rads_nofield = -1
! Math constants
real(eightbytereal), parameter :: pi = 3.1415926535897932d0, rad = pi/180d0
! I/O parameters
integer, parameter :: stderr = 0, stdin = 5, stdout = 6

! * Variables
! I/O variables
integer(fourbyteint), save :: rads_verbose = 0       ! Verbosity level
integer(fourbyteint), save :: rads_log_unit = stdout ! Unit number for statistics logging

! * RADS4 variable structures
type :: rads_varinfo                                 ! Information on variable used by RADS
```

```
    character(len=rads_varl) :: name               ! Short name of variable used by RADS
    character(len=rads_naml) :: long_name          ! Long name (description) of variable
    character(len=rads_naml) :: standard_name      ! Optional CF 'standard' name ('' if none)
    character(len=rads_naml) :: source             ! Optional data source ('' if none)
    character(len=rads_naml) :: parameters         ! Optional link to model parameters ('' if none)
    character(len=rads_strl) :: dataname           ! Name associated with data (e.g. netCDF var name)
    character(len=rads_cmdl) :: flag_meanings      ! Optional meaning of flag values ('' if none)
    character(len=rads_cmdl) :: quality_flag       ! Quality flag(s) associated with var ('' if none)
    character(len=rads_cmdl) :: comment            ! Optional comment ('' if none)
    character(len=rads_varl) :: units              ! Optional units of variable ('' if none)
    character(len=rads_varl) :: format             ! Fortran format for output
    character(len=rads_varl) :: gridx, gridy       ! RADS variable names of the grid x and y coords
    type(grid), pointer :: grid                    ! Pointer to grid (if data source is grid)
    real(eightbytereal) :: default                 ! Optional default value (Inf if not set)
    real(eightbytereal) :: limits(2)               ! Lower and upper limit for editing
    real(eightbytereal) :: plot_range(2)           ! Suggested range for plotting
    real(eightbytereal) :: add_offset, scale_factor  ! Offset and scale factor in case of netCDF
    real(eightbytereal) :: xmin, xmax, mean, sum2  ! Minimum, maximum, mean, sum squared deviation
    logical :: boz_format                          ! Format starts with B, O or Z.
    integer(fourbyteint) :: ndims                  ! Number of dimensions of variable
    integer(fourbyteint) :: brid                   ! Branch ID (default 1)
    integer(fourbyteint) :: nctype, varid          ! netCDF data type (nf90_int, etc.) and var ID
    integer(fourbyteint) :: datatype               ! Type of data (one of rads_type_*)
    integer(fourbyteint) :: datasrc                ! Retrieval source (one of rads_src_*)
    integer(fourbyteint) :: cycle, pass            ! Last processed cycle and pass
    integer(fourbyteint) :: selected, rejected     ! Number of selected or rejected measurements
endtype

type :: rads_var                                   ! Information on variable or alias
    character(len=rads_varl), pointer :: name      ! Pointer to short name of variable (or alias)
    character(len=rads_naml), pointer :: long_name ! Pointer to long name (description) of variable
    type(rads_varinfo), pointer :: info, inf1, inf2  ! Links to structs of type(rads_varinfo)
    logical(twobyteint) :: noedit                  ! .true. if editing is suspended
    integer(twobyteint) :: field(2)                ! RADS3 field numbers (rads_nofield = none)
endtype

type :: rads_cyclist                               ! List of cycles
    integer(fourbyteint) :: n, i                   ! Number of elements in list, additional value
    integer(fourbyteint) :: list(rads_cyclistl)    ! List of values
endtype

type :: rads_phase                                 ! Information about altimeter mission phase
    character(len=rads_varl) :: name, mission      ! Name (1-letter), and mission description
    integer(fourbyteint) :: cycles(2), passes      ! Cycle range and maximum number of passes
    real(eightbytereal) :: start_time, end_time    ! Start time and end time of this phase
    real(eightbytereal) :: ref_time, ref_lon       ! Time and lon of equator crossing of "ref. pass"
    integer(fourbyteint) :: ref_cycle, ref_pass    ! Cycle and pass number of "reference pass"
    real(eightbytereal) :: pass_seconds            ! Length of pass in seconds
    real(eightbytereal) :: repeat_days             ! Length of repeat period in days
    real(eightbytereal) :: repeat_shift            ! Eastward shift of track pattern for near repeats
    integer(fourbyteint) :: repeat_nodal           ! Length of repeat period in nodal days
    integer(fourbyteint) :: repeat_passes          ! Number of passes per repeat period
    type(rads_cyclist), pointer :: subcycles       ! Subcycle definition (if requested)
endtype

type :: rads_sat                                   ! Information on altimeter mission
    character(len=rads_naml) :: userroot           ! Root directory of current user (i.e. $HOME)
    character(len=rads_naml) :: dataroot           ! Root directory of RADS data (i.e. $RADSDATAROOT)
    character(len=rads_varl) :: branch(rads_max_branches) ! Name of optional branches
    character(len=rads_varl) :: spec               ! Temporary holding space for satellite specs
    character(len=rads_cmdl) :: command            ! Command line
    character(len=rads_naml), pointer :: glob_att(:) ! Global attributes
    character(len=8) :: satellite                  ! Satellite name
    real(eightbytereal) :: dt1hz                   ! "1 Hz" sampling interval
    real(eightbytereal) :: frequency(2)            ! Frequency (GHz) of primary and secondary channel
    real(eightbytereal) :: inclination             ! Satellite inclination (deg)
    real(eightbytereal) :: eqlonlim(0:1,2)         ! Equator lon limits for asc. and desc. passes
    real(eightbytereal) :: centroid(3)             ! Lon, lat, distance (in rad) selection criteria
    real(eightbytereal) :: xover_params(2)         ! Crossover parameters used in radsxoconv
    integer(fourbyteint) :: cycles(3),passes(3)    ! Cycle and pass limits and steps
    integer(fourbyteint) :: error                  ! Error code (positive = fatal, negative = warning)
    integer(fourbyteint) :: pass_stat(7)           ! Stats of rejection at start of rads_open_pass
    integer(fourbyteint) :: total_read, total_inside ! Total nr of measurements read and inside region
    integer(fourbyteint) :: nvar, nsel             ! Nr of available and selected vars and aliases
    logical :: n_hz_output                         ! Produce multi-Hz output
    character(len=2) :: sat                        ! 2-Letter satellite abbreviation
    integer(twobyteint) :: satid                   ! Numerical satellite identifier
    type(rads_cyclist), pointer :: excl_cycles     ! Excluded cycles (if requested)
    type(rads_var), pointer :: var(:)                ! List of available variables and aliases
    type(rads_var), pointer :: sel(:)                ! List of selected variables and aliases
    type(rads_var), pointer :: time, lat, lon        ! Pointers to time, lat, lon variables
    type(rads_phase), pointer :: phases(:)           ! Definitions of all mission phases
    type(rads_phase), pointer :: phase               ! Pointer to current phase
endtype

type :: rads_file                                    ! Information on RADS data file
    integer(fourbyteint) :: ncid                     ! NetCDF ID of pass file
    character(len=rads_cmdl) :: name                 ! Name of the netCDF pass file
endtype

type :: rads_pass                                    ! Pass structure
    character(len=rads_strl) :: original             ! Name of the original (GDR) pass file(s)
    character(len=rads_hstl), pointer :: history     ! File creation history
    real(eightbytereal) :: equator_time, equator_lon ! Equator time and longitude
    real(eightbytereal) :: start_time, end_time      ! Start and end time of pass
    real(eightbytereal), pointer :: tll(:,:)         ! Time, lat, lon matrix
    integer(twobyteint), pointer :: flags(:)         ! Array of engineering flags
    logical :: rw                                    ! NetCDF file opened for read/write
    integer(fourbyteint) :: cycle, pass              ! Cycle and pass number
    integer(fourbyteint) :: nlogs                    ! Number of RADS3 log entries
    integer(fourbyteint) :: ndata                    ! Number of data points (1-Hz)
    integer(fourbyteint) :: n_hz, n_wvf              ! Size second/third dimension (0=none)
    integer(fourbyteint) :: first_meas, last_meas    ! Index of first and last point in region
    integer(fourbyteint) :: time_dims                ! Dimensions of time/lat/lon stored
    integer(fourbyteint) :: trkid                    ! Numerical track identifiers
    type (rads_file) :: fileinfo(rads_max_branches)  ! File information for pass files
    type (rads_sat), pointer :: S                    ! Pointer to satellite/mission structure
    type (rads_pass), pointer :: next                ! Pointer to next pass in linked list
endtype

type :: rads_option                                  ! Information on command line options
    character(len=rads_varl) :: opt                  ! Option (without the - or --)
    character(len=rads_cmdl) :: arg                  ! Option argument
    integer :: id                                    ! Identifier in form 10*nsat + i
endtype

! These command line options can be accessed by RADS programs
type(rads_option), allocatable, target, save :: &
    rads_opt(:)                                      ! List of command line options
integer(fourbyteint), save :: rads_nopt = 0          ! Number of command line options saved
```

PURPOSE:

```
This module provides the main functionalities for the RADS4 software.
To use any of the following subroutines and functions, add the following
line in your Fortran 90 (or later) code:
  use rads
```

COPYRIGHT:

6.1.2 rads_init

SUMMARY:

```
Initialize RADS4
```

SYNTAX:

```
subroutine rads_init (S, sat, xml)
type(rads_sat), intent(inout) :: S <or> S(:)
character(len=*), intent(in), optional :: sat <or> sat(:)
character(len=*), intent(in), optional :: xml(:)
```

PURPOSE:

```
This routine initializes the <S> struct with the information pertaining
to given satellite/mission phase <sat>, which is to be formed as 'e1',
or 'e1g', or 'e1/g'. If no phase is specified, all mission phases will be
queried.

The <S> and <sat> arguments can either a single element or an array. In the
latter case, one <S> struct will be initialized for each <sat>.
To parse command line options after this, use rads_parse_cmd.

Only if the <sat> argument is omitted, then the routine will parse
the command line for arguments in the form:
--sat=<sat> --cycle=<lo>,<hi>,<step> --pass=<lo>,<hi>,<step>
--lim:<var>=<lo>,<hi> --lat=<lo>,<hi> --lon=<lo>,<hi> --alias:<var>=<var>
--opt:<value>=<value> --opt=<value>,... --fmt:<var>=<value>
or their equivalents without the = or : separators after the long name,
or their equivalents without the initial --, or the short options -S, -C,
-P, -L, -F

The routine will read the satellite/mission specific setup XML files and
store all the information in the stuct <S>. The XML files polled are:
  $RADSDATAROOT/conf/rads.xml
  ~/.rads/rads.xml
  rads.xml
  <xml> (from the optional array of file names)

If more than one -S option is given, then all further options following
this argument until the next -S option, plus all options prior to the
first -S option will pertain to this mission.

Execution will be halted when the dimension of <S> is insufficient to
store information of multiple missions, or when required XML files are
missing.

The verbosity level can be controlled by setting rads_verbose before
calling this routine (default = 0). The output unit for log info can
be controlled by setting rads_log_unit up front (default = stdout).
```

ARGUMENTS:

```
S        : Satellite/mission dependent structure
sat      : (optional) Satellite/mission abbreviation
xml      : (optional) Array of names of additional XML files to be loaded
```

6.1.3 rads_end

SUMMARY:

```
End RADS4
```

SYNTAX:

```
subroutine rads_end (S)
type(rads_sat), intent(inout) :: S <or> S(:)
```

PURPOSE:

```
This routine ends RADS by freeing up all <S> space and other allocated
global arrays.
```

ARGUMENT:

```
S        : Satellite/mission dependent struct or array of structs
```

6.1.4 rads_get_var

SUMMARY:

```
Read variable (data) from RADS4 file
```

SYNTAX:

```
recursive subroutine rads_get_var (S, P, var, data, noedit)
type(rads_sat), intent(inout) :: S
type(rads_pass), intent(inout) :: P
character(len=*) :: var
<or> integer(fourbyteint) :: var
<or> type(rads_var), intent(in) :: var
real(eightbytereal), intent(out) :: data(:)
logical, intent(in), optional :: noedit
```

PURPOSE:

```
This routine loads the data from a single variable <var> into the
buffer <data>. This command must be preceeded by <rads_open_pass>.
The variable <var> can be addressed as a variable name, a RADS3-type
field number or a varlist item.

The array <data> must be at the correct size to contain the entire
pass of data, i.e., it must have the dimension P%ndata.
If no data are available and no default value and no secondary aliases
then NaN is returned in the array <data>.
```

ARGUMENTS:

```
S         : Satellite/mission dependent structure
P         : Pass dependent structure
var       : (string) Name of the variable to be read.
                     If <var> ends with % editing is skipped.
            (integer) Field number.
            (type(rads_var)) Variable struct (e.g. S%sel(i))
data      : Data returned by this routine
noedit    : (optional) Set to .true. to skip editing on limits and/or
            quality flags; set to .false. to allow editing (default)
```

ERROR CODE:

```
S%error   : rads_noerr, rads_err_var, rads_err_memory, rads_err_source
```

6.1.5 rads_stat

SUMMARY:

```
Print the RADS statistics for a given satellite
```

SYNTAX:

```
subroutine rads_stat (S)
type(rads_sat), intent(in) :: S <or> S(:)
integer(fourbyteint), intent(in), optional :: unit
```

PURPOSE:

```
This routine prints out the statistics of all variables that were
processed per mission (indicated by scalar or array <S>), to the output
on unit <rads_log_unit>.
```

ARGUMENTS:

```
S         : Satellite/mission dependent structure
```

6.1.6 rads_set_options

SUMMARY:

```
Specify the list of command specific options

SYNPOSIS
```

PURPOSE:

```
Add the command specific options to the list of common RADS options.
The argument <optlist> needs to have the same format as in the routine
<getopt> in the <rads_misc> module. The short options will be placed
before the common ones, the long options will be placed after them.
```

ARGUMENT:

```
optlist  : (optional) list of command specific short and long options
```

6.1.7 rads_open_pass

SUMMARY:

```
Open RADS pass file
```

SYNOPSIS:

```
subroutine rads_open_pass (S, P, cycle, pass, rw)
use netcdf
use rads_netcdf
use rads_time
use rads_misc
use rads_geo
type(rads_sat), intent(inout) :: S
type(rads_pass), intent(inout) :: P
integer(fourbyteint), intent(in) :: cycle, pass
logical, intent(in), optional :: rw
```

PURPOSE:

```
This routine opens a netCDF file for access to the RADS machinery.
However, prior to opening the file, three tests are performed to speed
up data selection:
(1) All passes outside the preset cycle and pass limits are rejected.
(2) Based on the time of the reference pass, the length of the repeat
    cycle and the number of passes per cycle, a rough estimate is
    made of the temporal extent of the pass. If this is outside the
    selected time window, then the pass is rejected.
(3) Based on the equator longitude and the pass number of the reference
    pass, the length of the repeat cycle and the number of passes in
    the repeat cycle, an estimate is made of the equator longitude of
    the current pass. If this is outside the limits set in S%eqlonlim
    then the pass is rejected.

If the pass is rejected based on the above critetia or when no netCDF
file exists, S%error returns the warning value rads_warn_nc_file.
If the file cannot be read properly, rads_err_nc_parse is returned.
Also, in both cases, P%ndata will be set to zero.

By default the file is opened for reading only. Specify perm=nf90_write to
open for reading and writing.
The file opened with this routine should be closed by using rads_close_pass.
```

ARGUMENTS:

```
S         : Satellite/mission dependent structure
P         : Pass structure
cycle     : Cycle number
pass      : Pass number
rw        : (optional) Set read/write permission (def: read only)
```

ERROR CODE:

```
 S%error  : rads_noerr, rads_warn_nc_file, rads_err_nc_parse
```

6.1.8 rads_close_pass

SUMMARY:

```
 Close RADS pass file
```

SYNOPSIS:

```
subroutine rads_close_pass (S, P, keep)
use netcdf
use rads_netcdf
type(rads_sat), intent(inout) :: S
type(rads_pass), intent(inout) :: P
logical, intent(in), optional :: keep
```

PURPOSE:

```
 This routine closes a netCDF file previously opened by rads_open_pass.
 The routine will reset the ncid element of the <P> structure to
 indicate that the passfile is closed.
 If <keep> is set to .true., then the time, lat, lon and flags elements of
 the <P> structure are kept. Otherwise, they are deallocated along with
 the log entries.
 A second call to rads_close_pass without the keep argment can subsequently
 deallocate the time, lat and lon elements of the <P> structure.
```

ARGUMENTS:

```
 S        : Satellite/mission dependent structure
 P        : Pass structure
 keep     : Keep the P%tll matrix (destroy by default)
```

ERROR CODE:

```
 S%error  : rads_noerr, rads_err_nc_close
```

6.1.9 rads_read_xml

SUMMARY:

```
 Read RADS4 XML file
```

SYNOPSIS:

```
subroutine rads_read_xml (S, filename)
use netcdf
use xmlparse
use rads_time
use rads_misc
type(rads_sat), intent(inout) :: S
character(len=*), intent(in) :: filename
```

PURPOSE:

```
 This routine parses a RADS4 XML file and fills the <S> struct with
 information pertaining to the given satellite and all variable info
 encountered in that file.

 The execution terminates on any error, and also on any warning if
 fatal = .true.
```

ARGUMENTS:

```
 S        : Satellite/mission dependent structure
 filename : XML file name
 fatal    : If .true., then all warnings are fatal.
```

ERROR CODE:

```
 S%error  : rads_noerr, rads_err_xml_parse, rads_err_xml_file
```

6.1.10 rads_set_alias

SUMMARY:

```
 Set alias to an already defined variable
```

SYNOPSIS:

```
subroutine rads_set_alias (S, alias, varname, field)
use rads_misc
type(rads_sat), intent(inout) :: S
character(len=*), intent(in) :: alias, varname
integer(twobyteint), intent(in), optional :: field(2)
```

PURPOSE:

```
 This routine defines an alias to an existing variable, or up to three
 variables. When more than one variable is given as target, they will
 be addressed one after the other.
 If alias is already defined as an alias or variable, it will be overruled.
 The alias will need to point to an already existing variable or alias.
 Up to three variables can be specified, separated by spaces or commas.
```

ARGUMENTS:

```
 S        : Satellite/mission dependent structure
 alias    : New alias for (an) existing variable(s)
 varname  : Existing variable name(s)
 field    : (optional) new field numbers to associate with alias
```

ERROR CODE:

```
 S%error  : rads_noerr, rads_err_alias, rads_err_var
```

6.1.11 rads_set_limits

SUMMARY:

```
 Set limits on given variable
```

SYNOPSIS:

```
subroutine rads_set_limits (S, varname, lo, hi, string, iostat)
use rads_misc
type(rads_sat), intent(inout) :: S
character(len=*), intent(in) :: varname
real(eightbytereal), intent(in), optional :: lo, hi
character(len=*), intent(in), optional :: string
integer(fourbyteint), intent(out), optional :: iostat
```

PURPOSE:

```
 This routine set the lower and upper limits for a given variable in
 RADS.
 The limits can either be set by giving the lower and upper limits
 as double floats <lo> and <hi> or as a character string <string> which
 contains the two numbers separated by whitespace, a comma or a slash.
 In case only one number is given, only the lower or higher bound
 (following the separator) is set, the other value is left unchanged.
```

ARGUMENTS:

```
 S        : Satellite/mission dependent structure
 varname  : Variable name
 lo, hi   : Lower and upper limit
 string   : String of up to two values, with separating whitespace
            or comma or slash.
 iostat   : (optional) iostat code from reading string
```

ERROR CODE:

```
 S%error  : rads_noerr, rads_err_var
```

6.1.12 rads_set_region

SUMMARY:

```
 Set latitude/longitude limits or distance to point
```

SYNOPSIS:

```
subroutine rads_set_region (S, string)
use rads_misc
type(rads_sat), intent(inout) :: S
character(len=*), intent(in) :: string
```

PURPOSE:

```
 This routine set the region for data selection (after the -R option).
 The region can either be specified as a box by four values "W/E/S/N",
 or as a circular region by three values "E/N/radius". Separators
 can be commas, slashes, or whitespace.

 In case of a circular region, longitude and latitude limits are set
 accordingly for a rectangular box surrounding the circle. However, when
 reading pass data, the distance to the centroid is used as well to
 edit out data.
```

ARGUMENTS:

```
 S        : Satellite/mission dependent structure
 string   : String of three or four values with separating whitespace.
            For rectangular region: W/E/S/N.
            For circular region: E/N/radius (radius in degrees).
```

ERROR CODE:

```
 S%error  : rads_noerr, rads_err_var
```

6.1.13 rads_set_format

SUMMARY:

```
 Set print format for ASCII output of given variable
```

SYNOPSIS:

```
subroutine rads_set_format (S, varname, format)
type(rads_sat), intent(inout) :: S
character(len=*), intent(in) :: varname, format
```

PURPOSE:

```
 This routine set the FORTRAN format specifier of output of a given
```

```
variable in RADS.
```

ARGUMENTS:

```
S        : Satellite/mission dependent structure
varname  : Variable name
format   : FORTRAN format specifier (e.g. 'f10.3')
```

ERROR CODE:

```
S%error  : rads_noerr, rads_err_var
```

Bibliography

Francis, C. R. (1990), The ERS-1 radar altimeter, paper presented at the 2nd ERS-1 PI meeting, Noordwijk,The Netherlands.

Francis, C. R., et al. (1995), The ERS-2 spacecraft and its payload, *ESA Bulletin*, *83*, 13–31.

Francis, C. R., et al. (1991), The ERS-1 spacecraft and its payload, *ESA Bulletin*, *65*, 26–48.

Fu, L.-L., E. J. Christensen, C. A. Yamarone, M. Lefebvre, Y. Ménard, M. Dorrer, and P. Escudier (1994), TOPEX mission overview, *J. Geophys. Res.*, *99*(C12), 24,369–24,382.

Lambin, J., et al. (2010), The OSTM/Jason-2 mission, *Mar. Geod.*, *33*(S1), 4–25, doi:10.1080/01490419.2010.491030.

Ménard, Y., L.-L. Fu, P. Escudier, B. J. Haines, G. Kunstmann, F. Parisot, J. Perbos, P. Vincent, and S. D. Desai (2003), The Jason-1 mission, *J. Mar. Geod.*, *Jason-1*.

Wingham, D. J., et al. (2006), CryoSat: A mission to determine the fluctuations in Earth's land and marine ice fields, *Adv. Space Res.*, *37*(4), 841–871, doi:10.1016/j.asr.2005.07.027.

Index