
RADS version 4.1.2

User Manual

Remko Scharroo

13 October 2015

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

Contents

1	RADS software installation	1
2	RADS commands	2
3	RADS code	3
3.1	Module rads	3
3.1.1	rads	3
3.1.2	rads_init	3
3.1.3	rads_end	4
3.1.4	rads_get_var	5
3.1.5	rads_stat	5
3.1.6	rads_set_options	6
3.1.7	rads_open_pass	6
3.1.8	rads_close_pass	7
3.1.9	rads_read_xml	8
3.1.10	rads_set_alias	8
3.1.11	rads_set_limits	9
3.1.12	rads_set_region	10
3.1.13	rads_set_format	10
	Index	12

RADS software installation

Chapter 2

RADS commands

Chapter 3

RADS code

3.1 Module rads

3.1.1 rads

SUMMARY:

RADS main module

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:

Copyright (c) 2011-2015 Remko Scharroo
See LICENSE.TXT file for copying and redistribution conditions.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

3.1.2 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 'el', or 'elg', or 'el/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 struct <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
```

3.1.3 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

3.1.4 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=*) <or> integer(fourbyteint) <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 preceded 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

3.1.5 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

3.1.6 rads_set_options

SUMMARY:

Specify the list of command specific options

SYNOPSIS

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

3.1.7 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%eqnlim` then the pass is rejected.

If the pass is rejected based on the above criteria 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`

3.1.8 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` argument 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`

3.1.9 `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`

3.1.10 `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

```

3.1.11 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

```

3.1.12 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

```

3.1.13 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

Index

Functions

- rads_close_pass, 7
- rads_end, 4
- rads_get_var, 5
- rads_init, 3
- rads_open_pass, 6
- rads_read_xml, 8
- rads_set_alias, 8
- rads_set_format, 10
- rads_set_limits, 9
- rads_set_options, 6
- rads_set_region, 10
- rads_stat, 5

Modules

- rads, 3

unsorted

- rads, 3
- rads_close_pass, 7
- rads_end, 4
- rads_get_var, 5
- rads_init, 3
- rads_open_pass, 6
- rads_read_xml, 8
- rads_set_alias, 8
- rads_set_format, 10
- rads_set_limits, 9
- rads_set_options, 6
- rads_set_region, 10
- rads_stat, 5