Geneva Astronomical Data Centre

PARAM_GUI
USERS MANUAL

Reference	:	param_gui
Issue	:	1.2
Date	:	12 Mar 04

Geneva Astronomical Data Centre
Chemin d'Écogia 16
CH–1290 Versoix
Switzerland

http://isdc.unige.ch/index.cgi?Soft+astroroot

Authors and Approvals

AstroROOT	param_gui Users Manual	
12 Mar 04	1.2	param_gui 1.2

Prepared by : R. Rohlfs

Document Status Sheet

AstroROOT	param_gui Users Manual	
16 MAR 03	1.0	param_gui Users Manual released
29 MAY 03	1.1	New chapter: Application interface
12 MAR 04	1.2	The constructor ParamGui::ParamGui has a new parameter.
12 MAR 2004	Printed	

Contents

1 Overview

The param_gui program is a Graphical User Interface (GUI) to edit the parameters of a FTOOLS parameter file. The actual GUI depends on the content of the parameter file *.par. Additional GUI comments in the parameter file can define a more user friendly GUI. But it is always possible to use a standard FTOOLS parameter file to open a GUI window and to edit the parameters, to save them and to start the corresponding program.

Beside the param_gui program a library libparam_gui.a is available to call this GUI from an application program. Unfortunately the interface is not yet described. Please send an email to Reiner.Rohlfs@obs.unige.ch if you want to use this library.

2 How to start the param_gui program

After a successful installation (see http://isdc/index.cgi?Soft+astroroot) the program is installed in the bin sub-directory of the installation directory. The program has one mandatory and one optional parameter. The first program parameter defines the parameter file. It is not necessary to write the extension .par. The program will look for the parameter file in the directories defined by the environment variable PFILES. The program title bar shows the the full path of the used parameter file.

The second, the optional program parameter defines the program which can be started with the Run - button of the GUI. If it is not defined the GUI will start the program named as the parameter file but without extension .par and without path. It is assumed that the corresponding program is in one of the $PATH directories.

3 Hints for the User

- Move the mouse on top of the parameter name to see the parameter prompt in a tool tip (help balloon).
- The number entries for integer and real values have on their right side small buttons to increase and decrease the numerical value in the field. They also support using the up and down cursor keys to change the numerical values. The step size can be selected with control and shift keys:

  ```
  --                small step (1 unit/factor of 3)
  shift             medium step (10 units/factor of 10)
  control           large step (100 units/factor of 30)
  shift-control     huge step (1000 units/factor of 100)
  ```

  The steps are either linear or logarithmic. The default behavior is linear, but it can be changed by pressing the alt key at the same time.
- It is possible to select compresses FITS files (*.fits.gz) with the browse - button of text and file entries: Select first an other file filter in the file selection window before selecting a file. The filename without .gz is copied into the entry field of the file and string widget.
- If the entry field for file names or strings is too short to show the full text increase the width of the whole window. This will increase the width of the entry field as well.

4 Modifications of the Parameter File

The look and feel of the GUI is controlled by the parameter file of the associated executable.

First the standard parameter lines are interpreted and for each line a widget is created. The type of the widget depend on the data type of parameter. (See section 4.1 Widgets)

Beside that special comment lines modify the GUI. Each of these modification comment line **must** start at the beginning of the line with
GUI
These 5 characters including the space are the key for the GUI to find more information about a widget. Following this key one ore more fields are allowed. The fields are separated by a comma (,). Blanks and tabs just before and behind the comma are ignored.
The first field is always the parameter name. Note: there must be no comma between the key # GUI and the parameter name. Following fields define attributes for the widget of the given parameter. (See section 4.3 Attributes)
Here are some examples of these modification comment lines:

```
# GUI _FOLDER_MAIN,  X:2,  Y: 0.5, TABS: datasets | background | analysis mode
# GUI pointing-dol,  Z:datasets, DOL:2
# GUI pointing-dol,  X:4.3, Y:19, NAME:Pointing
# GUI _FRAME_ ,      X:1,  Y: 17, W:100, H:18, NAME:modes dependent definitions
# GUI image-err,     Z:output, DISABLE:mode=TIMING | SPECTRAL
# GUI image-sig,     Z:output, DISABLE:mode=TIMING | SPECTRAL
```

As shown above beside modification comment lines for parameters also modification lines which define "group widgets" like frames, folders and windows are allowed. For more information see section 4.1.2 Group Widgets.

A modification comment line for a parameter MUST be written in the parameter file after the parameter definition line!

4.1 Widgets

Each parameter in the parameter file builds one widget, called entry widget. Beside these widgets three further widgets can be build. These parameter independent widgets group several entry widgets and they are called group widgets. Frame, window and folder are implemented as group widgets.

4.1.1 Entry Widgets

- check box widget
  A check box widget is created for every boolean parameter (b). Mark the check box to set the boolean parameter to yes.
  Depending on the state of this widget other widgets can be disabled or enabled. See attribute DISABLE: in section 4.3 Attributes.
- integer entry widget
  An integer entry widgets is created for integer parameters (i) as long as no list of accepted values are defined in the min field of the parameter definition in the parameter file. See also section 3 Hints for the User.
- real entry widget
  An real entry widgets is created for real parameters (r) as long as no list of accepted values are defined in the min field of the parameter definition in the parameter file. See also section 3 Hints for the User.

- text entry widget
  A text entry widget is created for string (s) and for file (f) parameters as long as no list of accepted values are defined in the min field of the parameter definition in the parameter file. If the parameter is of type file (f) or if the attribute DOL: is defined for the parameter a "browse" button appears on the right side of the text entry field. With this button a file or a DOL can be selected, respectively.

- combo box widget
  A combo box is created for all parameters except boolean if a list of accepted values is defined in the min field of the parameter definition in the parameter file. As soon as the pipe character '|' is part of the min field a combo box is created. All in the min field listed and by '|' separated values are available for selection in the list box of the combo box.
  Depending on the selected item in the combo box widget other widgets can be disabled or enabled. See attribute DISABLE: in section 4.3 Attributes.

4.1.2 Group Widgets

Group widgets have in the first field of the # GUI comment line an identifier instead of the parameter name for entry widgets.

- Frame Widget
  A frame is just a rectangle area with a title (attribute NAME:). It is recommended to specify the width (attribute W:) and the height (attribute H:) of a frame.
  The identifier of a frame is **_FRAME_**

- Window Widget
  A window widget is a new window on the screen and an associated button in an other window. The user can open the new window by pressing this button. The attribute NAME: must be defined for a window widget! It is the Z - coordinate of the window and the text in the button. The coordinate attributes (X:, Y: and Z:) define the position of the associated button not of the window. To place a widget in this window the attribute Z:window_name must be defined for that widget.
  The identifier of a window is **_WINDOW_**

- Folder Widget
  A folder widget consist of several tabs. Each tab has a card with a name. This name is at the same time the Z - coordinate of the tab. At a given time only one tab with its widget is visible, while the other tabs are hidden "behind" this so called top - tab. The different tabs are defined with the TABS: - attribute which is required for this widget. To place a widget in one of the tabs the attribute Z:tab_name must be defined for that widget.
  The size of a folder widget depend on the size and position of the widgets in all the tabs of the folder. Therefore the attributes W: and H: cannot be defined for a folder widget.
  The identifier of a folder is **_FOLDER_**

To create more than one group widget of the same type or to define attributes for a group widget in more than one line in the parameter file the widget must have a unique identifier. Therefor it is possible to increase the widget specific identifiers by further characters. For example _FOLDER_name1 and _FOLDER_name2 are valid folder identifiers for two different folders.

4.2 The Coordinate System or the Layout

To place a widget in its window a three dimensional coordinate system is chosen. The X and Y coordinates are in units of a character size. (0/0) is at the upper left corner of a window. A folder has its own X/Y coordinate system where (0/0) is at the upper left corner of the tabs. The actual

coordinate values can be a fraction of a unit.
The third dimension, called Z, consist of different layers. Each layer has a name. A layer can be a window or a tab of a folder. The main window, for example has the Z - coordinate "main" and all hidden parameters are by default located in a window with a Z - coordinate named "hidden" . The Z - coordinate of a tab is the text on its card. To place a widget on a specific window or tab an attribute (Z:name) must be defined.

4.3 Attributes

Attributes are defined in a modification comment line (see section 4 Modifications of the Parameter File). Each attribute consist of an identifier and a value. The last character of an identifier is always a ':'. Attributes are separated by commas ',' in the parameter file. There is nearly always a default value for each attribute. Exceptions are clearly mentioned in the following description of each attribute. Note: not every attribute is recognize by each widget. See the table in section 4.4 Recognized - Attributes to verify which attributes can be used to modify a given widget.

- X:
  Defines the X position of the left edge of a widget on the window or on a tab. To use this value also the Y: - attribute must be defined for the same widget! If the Y: - attribute is not defined the default value is used.
  See also section 4.2 The Coordinate System or the Layout.

  The default value is 0.

- Y:
  Defines the Y position of the upper edge of a widget on the window or on a tab. To use this value also the X: - attribute must be defined for the same widget! If the X: - attribute is not defined the default value is used.
  See also section 4.2 The Coordinate System or the Layout.

  The default value is 2 more than the previous widget. The default value of the first widget is 0. Note: first the position of all widgets for which the X: - and Y: - attributes are defined are placed on the window or tab. Than the Y - position of the other widgets are determined by increasing the Y - coordinate by 2 starting with the largest predefined Y - position.

- Z:
  Defines the third dimension, the window or the tab of a folder, in which the widget will be visible. The value of this attribute therefor is not a number as for X: and Y: but a name. See section 4.2 The Coordinate System or the Layout for more information about the Z - dimension.

  The default value is main, which is the Z - coordinate of the first window.
  As exception the default value for hidden parameters is hidden, which is an other always available window. This 2nd window can be opened by a button of the main window.

- W:
  Defines the width of the widget. The parameter name in front of each widget is not included in this width.
  The width cannot be define for folders.

  The default value depend on the widget:

  - check box: 16 units
  - int entry: 14 units
  - real entry: 14 units
  - text entry: is either the global attribute TEXT_W or FILE_W for string and file parameters, respectively. If these global attributes are not define their default values are used (31 or 81 units). The button of the text entry widget is included in the width.

– combo box: 6 units more than the largest name of all entries in the combo box.

– frame: 16 units
– window: 10 units (this defines the width of the associated button to open the window)

- H:
  Defines the height of a widget. This attribute is recognized only by frames. The height of all other widgets is 2 units.
  The default value for frames is 6 units

- NAME:
  This attribute overwrites the parameter name on the GUI for entry widgets. It is the title for frame widgets and the Z - coordinate and the button text for window widgets. This attribute is required for windows!
  There is no default value.

- NAME_W:
  Defines the width of the parameter name or the value of the NAME: attribute on the screen. If the actual text is longer it will be overwritten by the entry widget. It the actual text is shorter than NAME_W: the entry widget will nevertheless keep space for the defined width. Therefor with this attribute the widgets can be aligned.
  The default value is define by the common attribute NAME_W which by itself has a default value of 16 units.

- TABS:
  This attribute is used only for folder widgets. It defines the names of all tabs and is a mandatory attribute. The names are separated by the pipe character |. Blanks and tabs just before and behind the | sign are ignored, but the tab names can include blanks. For example

```
# GUI _FOLDER_, TABS: global parameters | spectra parameters | image parameters
```

- DOL:
  This attribute is used for parameters which expect a DOL instead of a string or a filename. It is the exception of all attributes and is valid without specified value. But if a value is define like DOL:1, than this number is used as HDU position and automatically added to the selected file name. If no value is define like DOL:, than after a file is selected the GUI reads all data structures in the selected FITS file. The user has to select in a list box a HDU - name which is added to the selected file name.

- DISABLE:
  With this attribute widgets can be disabled depending on the status of a check box widget or depending on the selected item in a combo box widget. The value string of this widget has a specific syntax:

```
# example for combo box
# GUI param_name1, DISABLE: depend_on= image
# GUI param_name2, DISABLE: depend_on= image | timing
# example for check box
# GUI param_name3, DISABLE: depend_on1 = yes
# GUI param_name4, DISABLE: depend_on1 = no
```

  depend_on is the parameter name (not the attribute NAME:) of an other widget. This has to be either a check box widget or a combo box widget! If in the example above timing in the combo box depend_on is selected than the widget for param_name will be enables while the widget for param_name2 will be disabled.
  If disable depends on a check box widget the name behind the = sign has to be either yes or no! Any other string will be ignored.
  It is not possible to define a dependency from two different widgets for one parameter. The

second definition will overwrite the previous one.
By default a widget does not depend on the state of any other widget and is enabled all the time.

4.3.1 Common Attributes

Common attributes are used to define default values. These default values are applied for all parameters which are located behind the modification comment line of the common attribute in the parameter file. It is possible to reset the default values several times in a parameter file.
The identifier for common attributes is _COMMON_ for example:

```
# GUI _COMMON_,  NAME_W: 25, TEXT_W: 80
```

- NAME_W:
  Defines the default value for the NAME_W attribute of each individual widget.
  The default value for this common attribute is 16 units.
- TEXT_W:
  Defines the default value for the width of a text entry widget when it is created for a string parameter (s).
  The default value for this common attribute is 31 units.
- FILE_W:
  Defines the default value for the width of a text entry widget when it is created for a file parameter (f).
  The default value for this common attribute is 81 units.

4.4 Recognized - Attributes

Following table shows for each widget the attributes that are recognized by the widget. Defining an attribute for a widget which does not recognize this attribute is not a problem at all but has no affect at the GUI.

Widget	param type	X:	Y:	Z:	W:	H:	NAME:	NAME_W:	TABS:	DOL:	DISABLE:
check box	bool (b)	X	X	X	X		X	X			X
int entry	int (i)	X	X	X	X		X	X			X
real entry	real (r)	X	X	X	X		X	X			X
text entry	s or f	X	X	X	X		X	X		X	X
combo box	i, r, s or f	X	X	X	X		X	X			X
frame		X	X	X	X	X	X				
folder		X	X	X					required		
window		X	X	X	X		required				

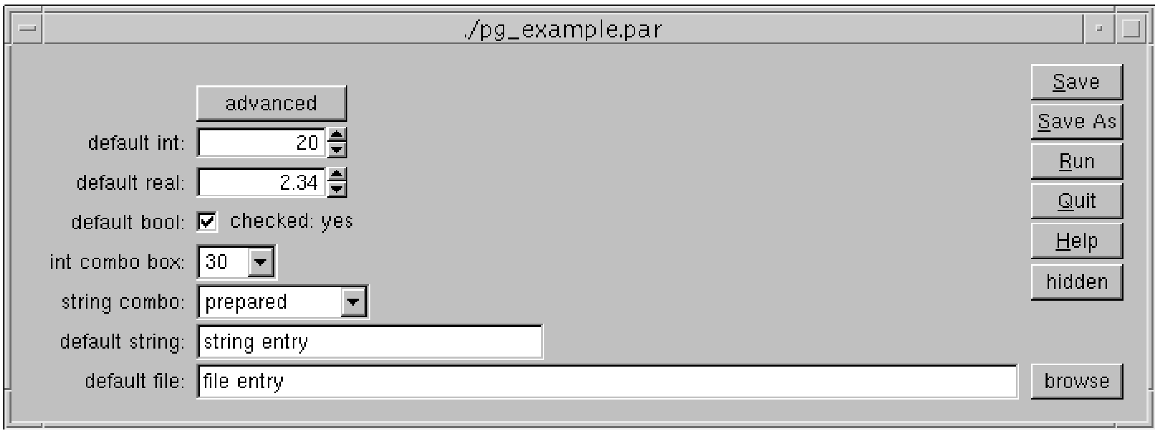
5 Example

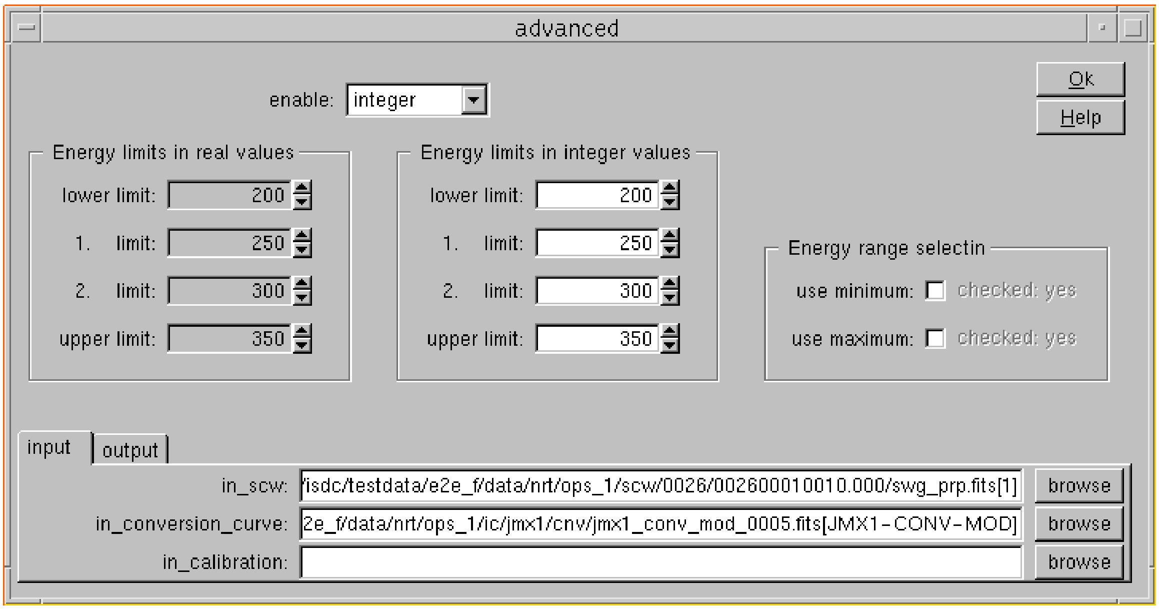
Following example parameter file creates a GUI with two windows. All widgets on the main window use default attributes and no attribute at all is defined for these parameters with the exception of the "advanced" - button.
The second window is an example showing all group widgets and a more user friendly GUI. Attributes are defined for each widget to define the size and the position. Selecting an entry in the "enable" combo box enables and disables the widgets located in the frames.

```
# first an example of every data type without any attribute

default int, i, q, , 20, 40, integer entry without any attributes
default real, r, l, , 2.34, 4.56, real entry without any attribute
default bool, b, l,yes,,, boolean entry without any attribute
int combo box, i, l,30, 20|30|40|50|60,,combo box of integers
string combo,s, ql,,raw|prepared|calibrated|,, combo box of strings
default string, s, l, string entry,,,string entry without any attribut
default file, f, l, file entry,,,file entry without any attribute

# sedond a more advanced GUI

# GUI _WINDOW_AD, NAME: advanced, X:15, Y: 2, W:15

# GUI _COMMON_, NAME_W: 25

enable,s,h,integer,integer| real | boolean,, enable / disable widgets
# GUI enable, Z: advanced,  X: 5, Y: 2

# GUI _COMMON_, NAME_W: 12
min_eng, r, l, , 200, 400,Define lower energy limit
l1_eng,  r, l, , 250, 450,Define 1. enery limit
l2_eng,  r, l, , 300, 500,Define 2. enery limit
max_eng, r, l, , 350, 550,Define upper. enery limit

# GUI min_eng, Z: advanced, X: 2, Y: 7,    NAME: lower limit, DISABLE:enable=integer|boolean
# GUI l1_eng,  Z: advanced, X: 2, Y: 9.5,  NAME: 1.    limit, DISABLE:enable=integer|boolean
# GUI l2_eng,  Z: advanced, X: 2, Y: 12,   NAME: 2.    limit, DISABLE:enable=integer|boolean
```

```
# GUI max_eng, Z: advanced, X: 2, Y: 14.5, NAME: upper limit, DISABLE:enable=integer|boolean
# GUI _FRAME_ENG, Z: advanced, X:1, Y:5, W:30, H: 13, NAME: Energy limits in real values

min_eng_i, i, l, , 200, 400,Define lower energy limit
l1_eng_i,  i, l, , 250, 450,Define 1. enery limit
l2_eng_i,  i, l, , 300, 500,Define 2. enery limit
max_eng_i, i, l, , 350, 550,Define upper. enery limit

# GUI min_eng_i, Z: advanced, X: 35, Y: 7,    NAME: lower limit, DISABLE:enable=real|boolean
# GUI l1_eng_i,  Z: advanced, X: 35, Y: 9.5,  NAME: 1.    limit, DISABLE:enable=real|boolean
# GUI l2_eng_i,  Z: advanced, X: 35, Y: 12,   NAME: 2.    limit, DISABLE:enable=real|boolean
# GUI max_eng_i, Z: advanced, X: 35, Y: 14.5, NAME: upper limit, DISABLE:enable=real|boolean
# GUI _FRAME_ENG_I, Z: advanced, X:34, Y:5, W:30, H: 13, NAME: Energy limits in integer values

# GUI _COMMON_, NAME_W: 14
min_eng_b, b, l, yes,,,Use lower range
max_eng_b, b, l, no,,,Use upper range
# GUI min_eng_b, Z: advanced, X: 68, Y: 12,   NAME: use minimum, DISABLE:enable=real|integer
# GUI max_eng_b, Z: advanced, X: 68, Y: 14.5, NAME: use maximum, DISABLE:enable=real|integer
# GUI _FRAME_ENG_B, Z: advanced, X:67, Y:10, W:32, H: 8, NAME: Energy range selectin

# GUI _COMMON_, NAME_W: 25, TEXT_W:75, FILE_W:75
# GUI _FOLDER_, Z: advanced, X: 0, Y: 20, TABS:input|output
in_scw,s,l,,,,input science window
in_conversion_curve,s,l,,,,input conversion curve
in_calibration,s,l,,,,input calibratin table
# GUI in_scw, Z:input, DOL:1
# GUI in_conversion_curve, Z:input, DOL:IBIS-CONV-MOD
# GUI in_calibration, Z:input, DOL:3

out_result,f,l,,,,output of calculations
# GUI out_result, Z:output, X:0, Y: 2, DOL:
```

These are the two windows created with the parameter file of the previous pages.





6 Application Interface

The libparam_gui.a library includes the class **ParamGui** which can be used by applications to open a GUI with all its parameters and to get the values of the parameters defined by the user:

```
class ParamGui
{
public:
   ParamGui(int argc, char ** argv, TApplication * app = NULL);

   bool           Quit();
   int            Error();

   bool           GetBool(const char * paraName);
   int            GetInt(const char * paraName);
   double         GetReal(const char * paraName);
   const char *   GetString(const char * paraName);
};
```

The file param_api.h has to be included to be able to use this class.

The constructor opens the GUI of the program and does not return until the user pressed the Quit or the Run button of the GUI. The parameter app can be set to and already created TApplication. If it is set to NULL (default value) the function will create a TApplication by itself. Note: There should be never two instances of TApplication in one program. Make sure that an already created TApplication is passed to the constructor of ParamGui to avoid creating a second instance.

The Quit() method returns true if the user pressed the Quit button and returns false if the user pressed the Run button. If the user pressed the Run button (Quit() returned false) the application can asked for the values of the parameters with the Get*() - methods. The GetString() - method should be used for strings and file - parameters.

The method Error() should return 0 to indicate that no error occurred.
-1: the parameter file could not be opened.
-2: the requested parameter of one of the Get*() - methods does not exist.
-30xx: a PIL error. See PIL user manual.

Following example demonstrates the use of this class:

```
#include "param_api.h"

main(int argc, char ** argv)
{
   ParamGui gui(argc, argv);  // opens the GUI

   if (gui.Error() != 0)
      {
      printf("Error: %d\n", gui.Error());
      return -1;
      }

   if (gui.Quit())
      return 0;         // the user pressed the Quit button

   // we can ask for the values of the parameters

   int          ii  = gui.GetInt("default int");
   double       dd  = gui.GetReal("default real");
   const char * str = gui.GetString("default string");
   bool         bb  = gui.GetBool("default bool");

   printf("error: %d\n", (int)gui.Error());
   printf("int: %d  real: %12.4f   string: %s   bool : %d\n",
          ii, dd, str, bb);
}
```

7 Errors and Warnings

- **Selected files does not has any HDU or is no FITS file!**
  If the DOL: attribute is defined without a DOL-value the GUI reads all HDU - names after a file is selected by the user. This error message appears if no HDU can be opened with a DAL function.
- **Environment variable ISDC_ENV is not defined**
  Cannot find the help file!

When the user pressed the Help - button the GUI tries to find the help file of the corresponding script or executable in the ${ISDC_ENV}/help directory.

- **File `filename` is missing.**
  Cannot display help text.
  The environment variable ISDC_ENV is set but the help text of the script of executable is not available in the directory ${ISDC_ENV}/help

- **Cannot open file**
  `filename`
  Cannot display help text
  The help text is available in the ${ISDC_ENV}/help directory, but the system call to open the file failed. Check the read permission of the help file.