

SACLANTCEN SPECIAL REPORT
serial no.: M-127

**SACLANT UNDERSEA
RESEARCH CENTRE
SPECIAL REPORT**



RABEL
A PROGRAM TO REPLAY ABEL
VERSION 2.2

G. Canepa

May 1997

The SACLANT Undersea Research Centre provides the Supreme Allied Commander Atlantic (SACLANT) with scientific and technical assistance under the terms of its NATO charter, which entered into force on 1 February 1963. Without prejudice to this main task – and under the policy direction of SACLANT – the Centre also renders scientific and technical assistance to the individual NATO nations.

This document is approved for public release.
Distribution is unlimited

SACLANT Undersea Research Centre
Viale San Bartolomeo 400
19138 San Bartolomeo (SP), Italy

tel: +39-187-540.111
fax: +39-187-524.600

e-mail: library@saclantc.nato.int

NORTH ATLANTIC TREATY ORGANIZATION

RAbel

A Program to Replay Abel

Version 2.2

Gaetano Canepa

The content of this document pertains to work performed under Project 033-2 of the SACLANTCEN Programme of Work. The document has been approved for release by The Director, SACLANTCEN.



Jan L. Spoelstra
Director

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

RAbel

A Program to Replay Abel

Version 2.2

Gaetano Canepa

Abstract: The description of a graphics software tool used to view and manipulate data file is presented. The software is implemented to run in the IDL[®] environment on any platform. **RAbel** is able to read data from files in the **ADAM** format, a **SACLANTCEN** standard used to save data acquired using the **ABEL** system. The software has a sophisticated graphic user interface which simplifies loading and scanning of data files. The program is also able to perform simple operations on the data: enveloping, zooming, filtering, *etc.* This report is intended as a detailed description of the implementation at a source code level. A short introduction to the program capability is given.

Keywords: IDL routine, graphic packages, data file format, data analysis

NATO UNCLASSIFIED

SACLANTCEN M-127

Contents

1	Introduction	1
1.1	How to run the package	1
1.2	The makefile	3
1.3	Manual organization	5
1.4	Different data file versions	5
2	Program description	7
2.1	Program features	7
2.2	Function useable by the user	14
3	Main program: <code>rabel</code>	16
3.1	<code>rabel</code>	16
4	Data Management routines: <code>io_rabel</code>	18
4.1	<code>load_io_file</code>	18
4.2	<code>load_io_ni</code>	19
4.3	<code>get_data()</code>	23
4.4	<code>set_data</code>	24
4.5	<code>dataset()</code>	25
4.6	<code>xset()</code>	26
4.7	<code>datamat()</code>	27
4.8	<code>dataping()</code>	29
4.9	<code>n_of_ping()</code>	31
4.10	<code>set_ping</code>	31
4.11	<code>set_card_chan</code>	33
4.12	<code>is_file_loaded()</code>	33
4.13	<code>data_file()</code>	34
4.14	<code>setfile</code>	35
4.15	<code>get_header()</code>	35
4.16	<code>get_channel_head()</code>	36
4.17	<code>get_ping_head()</code>	36
4.18	<code>getr_ping_head()</code>	37
4.19	<code>get_version()</code>	38
4.20	<code>get_freq()</code>	39
4.21	<code>n_of_byte()</code>	40
4.22	<code>n_of_chan()</code>	40
4.23	<code>get_nochan()</code>	41
5	Menu managing routines: <code>m_rabel</code>	43
5.1	<code>make_m_menu</code>	44
5.2	<code>remake_m_menu</code>	45
5.3	<code>save_menu_bar</code>	47
5.4	<code>change_menu_bar</code>	47

NATO UNCLASSIFIED

6	Management of the main window: cm_rabel	49
6.1	Main window dimensions: cm_rabel.hpr	49
6.2	main_w_create	50
6.3	main_w_destroy	53
6.4	main_w_resize	53
6.5	cm_rabel	54
6.6	main_id()	56
6.7	quit_rabel	56
7	Event managing routine: e_rabel	58
7.1	MAIN_RABEL_Event	58
7.2	MENU_Event	59
8	Graphic routines: g_rabel	61
8.1	zoom_event	61
8.2	ping_slider_event	63
8.3	reset_zoom	64
8.4	interval()	64
8.5	pan	65
8.6	set_title	66
8.7	set_slider	67
8.8	redraw	67
9	Print widget: pl_rabel	71
9.1	print_graph	71
9.2	PRINT_GRAPH_Event	74
10	Routines that implement events: er_rabel	76
10.1	help_rabel	76
10.2	about_rabel	77
10.3	exit_to_id1	77
11	Filtering widget: f_rabel	79
11.1	filter_wid	79
11.2	FILTER_WID_Event	82
11.3	get_filter()	83
12	Data conversion routines: type_con	85
12.1	convert()	85
12.2	conv_head()	86
13	Status bar routines: sb_rabel	88
13.1	make_status_bar	88
13.2	set_range	89
13.3	SET_RANGE_Event	90
13.4	change_status_bar	91
13.5	destroy_status_bar	92
13.6	STATUS_BAR_Event	92
13.7	change_info	93
13.8	header_info	94
13.9	remove_info	96

NATO UNCLASSIFIED

SACLANTCEN M-127

13.10	transmitter_info	97
13.11	channel_info	99
13.12	change_channel_info	101
13.13	ping_info	102
13.14	change_ping_info	104
14	Analysis of the signal: op_rabel	107
14.1	create()	107
14.2	free	109
14.3	get_handle	110
14.4	energy	111
14.5	copy	112
14.6	a_energy	113
14.7	envelope	114
14.8	a_envelope	116
14.9	env()	117
14.10	mean()	117
14.11	spectrum	118
14.12	a_spectrum	119
14.13	image	121
14.14	image_abs	122
14.15	swell()	123
14.16	filter	125
15	Data calculation: ca_rabel	128
15.1	make_calculator	130
15.2	set_calc	132
15.3	ca_create	133
15.4	ca_recreate	134
15.5	ca_destroy	134
15.6	ca_swap	135
15.7	MAIN_CALC_Event	135
15.8	ca_push	137
15.9	ca_add	138
15.10	ca_delete	139
15.11	ca_copy	140
15.12	ca_exit	141
15.13	new_handle()	141
15.14	set_ptr	142
16	Adam file format specification, Version 2	144
16.1	Introduction	144
16.2	Data types and sizes	144
16.3	The adamID	145
16.4	The adamHead	146
16.5	The adamChanHead header	148
16.6	The adamPingHead header	149
16.7	Data block format	150
16.8	Tail	150
16.9	Sampling	151

NATO UNCLASSIFIED

16.10More information 151

Index 153

1

Introduction

This is the user manual of **RAbel**: an IDL[®] package used to replay data acquired using the **Abel** system.

Abel is a complete acquisition system (software plus hardware) used to deal with acoustic experiment. The software read the acoustic data acquired using an Analog to Digital converter and an EISA Digital card, the emitter position from various positioning devices (PUAPS and DGPS), and the motion reference unit (MRU) data. The data are saved using an internal standard format called **Adam**. **Adam**¹ is a special format used to store a great amount of data and the related information on a file.

Abel was built having in mind a hardware with a maximum of 2 DA cards (the 2.0 release of **Adam** deal with a maximum of 16 cards); each card has a maximum of 16 analog input channels. The data files usually contain more than one answer to an acoustic emission: each of these answers is called in the following “ping”.

1.1 How to run the package

To run this package all the package files in the release media must be copied into a directory the name of which must be appended to the `IDL_PATH`. Then, it is possible to run **RAbel** using, inside IDL[®], the command `rael`.

The files of the **RAbel** package are obtained applying the program `web2pro` to the `LATEX` files which create this manual. `web2pro` extracts the segment of program from the manual files. In the release medium there is a **Makefile** which automatically generates manual and program files starting from the package `LATEX` files. The `C` language version and the `WIN32` executable version of the `web2pro` program are also included in the package.

To run the package only the following `.pro`, `.hpr`, and `.com` files are necessary²:

¹The **Adam** format is described in detail in Section 16

²The `.com` file are not executable files but file there the IDL[®] common blocks are defined.

NATO UNCLASSIFIED

SACLANTCEN M-127

List of Figures

1	The appearance of the main RAbel window.	8
2	a) The contents of the main menu item File and b) an example of the widget used for loading file (Windows [®] environment).	9
3	a) An example of how to select channel 6 card 1 pings; b) an example of the widget used to load the color table (Windows [®] environment).	10
4	An example of the widgets showing the contents of the data file headers: the channel (a), and the main header headers (b).	11
5	An example of the widgets showing the contents of the data file headers: the ping header.	12
6	The operations menu item.	13
7	The appearance of the printing widget.	72
8	The appearance of the filter widget.	80
9	The results of an image operation using the swell filter.	121
10	The appearance of the calculator when only three data are in the stack.	129
11	The overall format of an Adam data file.	152

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

1. `rael.pro`, `rael.ico`: the main file for **RAbel** and the icon for the Windows[®] environment
2. `cm_rael.hpr`: the header initialization file for **RAbel**: it is possible to personalize it
3. `cm_rael.pro`, `cm_rael.com`: the constructor of the main window
4. `m_rael.pro`, `m_rael.com`: routines managing the main window menu bar
5. `m_rael.hpr`: the structure of the main window menu bar
6. `io_rael.pro`, `io_rael.com`: routines managing data file input/output
7. `io_rael.hpr`: a part of the structure of **Adam** file: standard for all the file versions
8. `io_rael.hpr`: a part of the structure of **Adam** file: features of version 1
9. `io_rabe2.hpr`: a part of the structure of **Adam** file: features of version 2
10. `e_rael.pro`: event handler routines
11. `g_rael.pro`: routines managing the re-sizable widgets and the drawing windows
12. `er_rael.pro`: routines implementing event actions which are of general use
13. `sb_rael.pro`: status bar routines
14. `cw_pdmer.pro`: a slightly modified version of the **CW_PDMENU** IDL[®] standard routine; small changes are not commented on.
15. `ca_rael.pro`, `ca_rael.ico`, `ca_rael.com`: implementation of the data calculator
16. `f_rael.pro`: filter options widget implementation
17. `pl_rael.pro`: printing widget implementation
18. `op_rael.pro`, `op_rael.com`: analysis routines
19. `cw_fslir.pro`: a slightly different version of the IDL[®] routine **CW_FSLIDER**; the small changes are not commented here
20. `type_con.pro`: routines used to convert data from hp and sun to alpha and vice-versa

NATO UNCLASSIFIED

All the files have short names to be used on DOS systems.

To place an icon on the Windows[®] workspace automatically starting RAbel if double clicked, use the `rael.ico` icons and run the command `idl32 runrael.runrael.pro` is a script file which automatically compiles and runs RAbel.

The RAbel manual is available in the package medium in two formats: a PostScript[®] printable format `.ps`, and a device independent `.dvi` format:

- `rael.dvi`: a dvi file containing the official manual of RAbel
- `rael.ps`: the PostScript[®] version of the official manual of RAbel

It is also possible to re-compile the manual using a L^AT_EX compiler.

1.2 The makefile

The following lines show the contents of the makefile used to produce the manual and the executable program of RAbel.

```
## Makefile for building program and manual of RAbel
## Created by G. Canepa
##

.SUFFIXES: .tex .pro .hte .hpr

PROGRAMS = e_rael.pro g_rael.pro m_rael.pro type_con.pro \
pl_rael.pro f_rael.pro er_rael.pro io_rael.pro rael.pro \
cm_rael.pro sb_rael.pro op_rael.pro ca_rael.pro

HEADERS = cm_rael.hpr m_rael.hpr

help :
@echo "\nThis makefile has the following options: \n"
@echo "make help:      This help"
@echo "make clean:     Remove all files except source"
@echo "make manual:    Create dvi version of RAbel manual"
@echo "make ps:        Create the postscript version of the manual"
@echo "make rael:      Make RAbel executable"
@echo "make all:       Make RAbel executable and dvi RAbel manual"
```

NATO UNCLASSIFIEDSACLANTCEN M-127

```
manual :  
latex rabelman  
  
ps :  
latex rabel  
dvips rabel  
  
rabel : $(PROGRAMS) $(HEADERS)  
  
e_rabel.pro : e_rabel.tex  
web2pro e_rabel.tex pro  
  
g_rabel.pro : g_rabel.tex  
web2pro g_rabel.tex pro  
  
m_rabel.pro : m_rabel.tex  
web2pro m_rabel.tex pro  
  
cm_rabel.pro : cm_rabel.tex  
web2pro cm_rabel.tex pro  
  
type_con.pro : type_con.tex  
web2pro type_con.tex pro  
  
er_rabel.pro : er_rabel.tex  
web2pro er_rabel.tex pro  
  
rabel.pro : rabel.tex  
web2pro rabel.tex pro  
  
io_rabel.pro : io_rabel.tex  
web2pro io_rabel.tex pro  
  
sb_rabel.pro : sb_rabel.tex  
web2pro sb_rabel.tex pro  
  
op_rabel.pro : op_rabel.tex  
web2pro op_rabel.tex pro  
  
pl_rabel.pro : pl_rabel.tex  
web2pro pl_rabel.tex pro  
  
f_rabel.pro : f_rabel.tex
```

NATO UNCLASSIFIED

```

web2pro f_rabel.tex pro

ca_rabel.pro : ca_rabel.tex
web2pro ca_rabel.tex pro

cm_rabel.hpr : cm_rabel.hte
web2pro cm_rabel.hte hpr

m_rabel.hpr : m_rabel.hte
web2pro m_rabel.hte hpr

all :
$(MAKE) rabel
$(MAKE) ps

clean :
rm $(PROGRAMS) $(HEADERS) *~ *.dvi *.log *.aux *.ps *.toc

```

1.3 Manual organization

This manual is embedded with the program implementation. In fact, it is possible to extract the program part from the manual files using the **web2pro** program (distributed with this package). This manual is organized as follows:

- Each file of routines is illustrated in a different section of the manual, except **rabel.tex**.
- Each subsection describes a function, a structure, or an important block of information using a standard information header.
- A table of contents and an index of the functions facilitate retrieval of information.
- The functions implemented in **RAbel** always have lower case name, to distinguish them from the IDL[®] built in functions, which always have uppercase names. Reserved IDL[®] keywords are uppercase.

1.4 Different data file versions

The format of the data file (**Adam**) is subject to change according to preceding file format errors or limits and on requirements due to new kind of instrumentation

NATO UNCLASSIFIEDSACLANTCEN M-127

and experimental setup. Because of these format changes, **RAbel** adapts to different file versions. When the data file is read for the first time the version number is read and a **RAbel** variable is set. A function which returns the **Adam** file version (**get_version()**) is also defined. To avoid the necessity of important change for each new data file version access to the data file headers is encapsulated. The use of **get_version()** was limited to two files: **io_rabel** and **sb_rabel**, due to the fact that these files will need many changes for each data file update. All the functions implemented in the other files access the data file contents exclusively by means of specific functions.

NATO UNCLASSIFIED

2

Program description

RAbel is used to load and manage data files stored using the **Adam** format. Figure 1 shows the main window of **RAbel** once a data file is loaded. To load the data file select the **Load** item in the **File** menu. Figure 2.a shows the contents of the main menu item **File**. The **Adam** data file is loaded and used (independently from the version number and from the hardware and file system type). If the file is loaded using the graphic user interface (GUI), a **Window**[®] or **X11**[®] widget is used to select the file (Fig. 2.b). If the **Adam** format is the 1.0 version, data acquired from a system with only two cards can be stored; if the **Adam** format is the 2.0 version, the number of usable cards is increased to 16. Each card has up to 16 channels and each file can contain a certain number of pings. Using **RAbel** it is possible to select the **card**, **channel**, and **ping** numbers of the signal to be shown. To select the card and channel (by GUI) click on the main **RAbel** menu item **Channel**, select one of the active cards and then one of the active channels (Fig. 3.a). To select the ping use the **Ping slider** placed between the two graphic windows in Fig. 1. It is possible to perform a series of operation on the data from more than one ping. It is therefore possible to select a range of pings, starting from the ping actually selected and including the **range** number of pings.

2.1 Program features

Printing drawing widgets Selecting the **Print** option in the **File** menu item, it is possible to print the two drawing widgets setting different options (see Section 9).

Different Adam file versions Different releases of the **Adam** file data structure are implemented (version number 1.0 and 2.0 to now).

Personal lay-out The screen layout may be resized changing the contents of a single header file (**cm_rabel.hpr**)³. To customize the configuration copy the **cm_rabel.hpr** file in a directory which is placed before the **RAbel** directory in the

³See Subsection 6.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

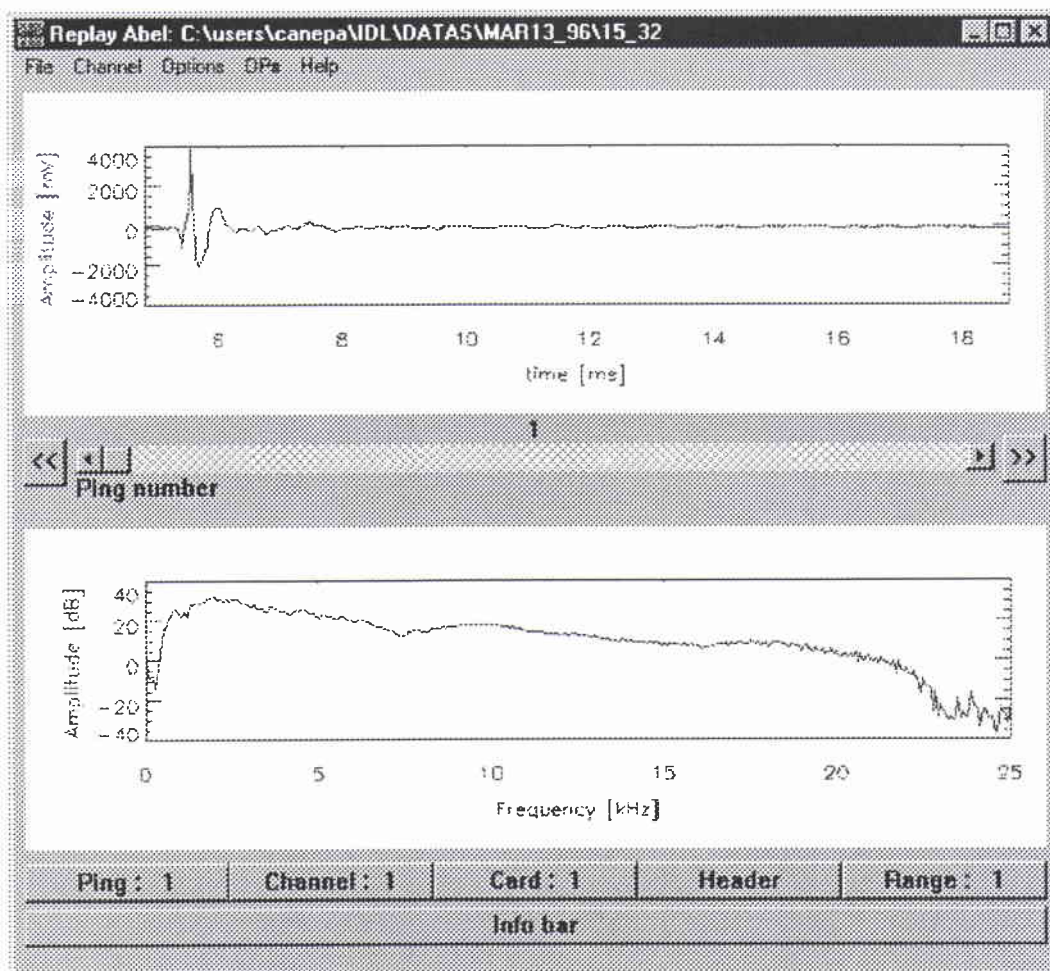


Figure 1: *The appearance of the main RAbel window.*

IDL_PATH environment variable following which it is possible to change the RAbel variables according to requirements (see Fig. 1).

Zooming It is possible to zoom the ping graph along the time axis by clicking the left mouse button at one point in the upper graph window, dragging the pointer to another point, and releasing the left mouse button. A new graph will be shown with an abscissa-axis varying between the two selected points. There is no limit to the number of zooms. To return to the normal view click and drag the mouse pointer in the right to left direction in the upper graphic window.

NATO UNCLASSIFIED

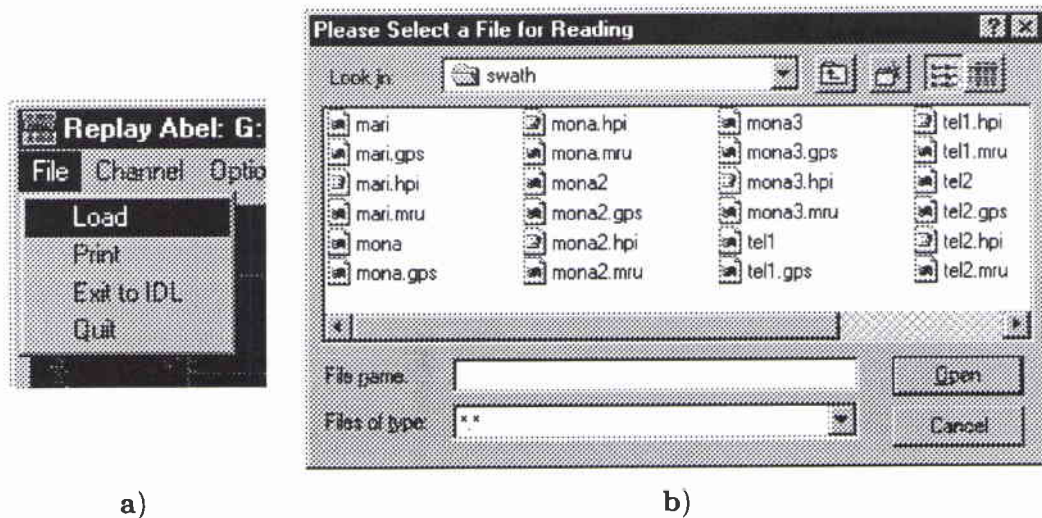


Figure 2: a) The contents of the main menu item *File* and b) an example of the widget used for loading file (Windows® environment).

Exit to IDL® The user can temporarily interrupt the **RAbel** session to work in IDL® retaining the use of the **RAbel** functions and, in particular, the functions which give abscissa and ordinate of the data (`xset()`, `dataset()`⁴, /etc.). Other useful routines permit the user to set or obtain the data ping, channel, and card numbers (`set_data()`, and `get_data()`).

As an example, the user can start **RAbel**, load a data file, select a given ping and channel and then can exit to IDL® and use the `xset()`, and `dataset()` functions to obtain two vectors containing the abscissa and ordinate of the selected signal. All the useful variables of **RAbel** are available to the user which can access them by means of functions.

In the following section of the **RAbel** manual all the **RAbel** functions are described; in the header of each function it is also specified if the function can be used out of **RAbel** or if this possibility is not supported or desired. Subsection 2.2 describes an index of the functions that can be used as external command in IDL®. Almost all the functions require that the **RAbel** main window is open.

Options At present, the only available option is the possibility to load a colour table using a simple interface given in the IDL® package (Fig. 3.b).

⁴See Section 4.

NATO UNCLASSIFIED

SACLANTCEN M-127

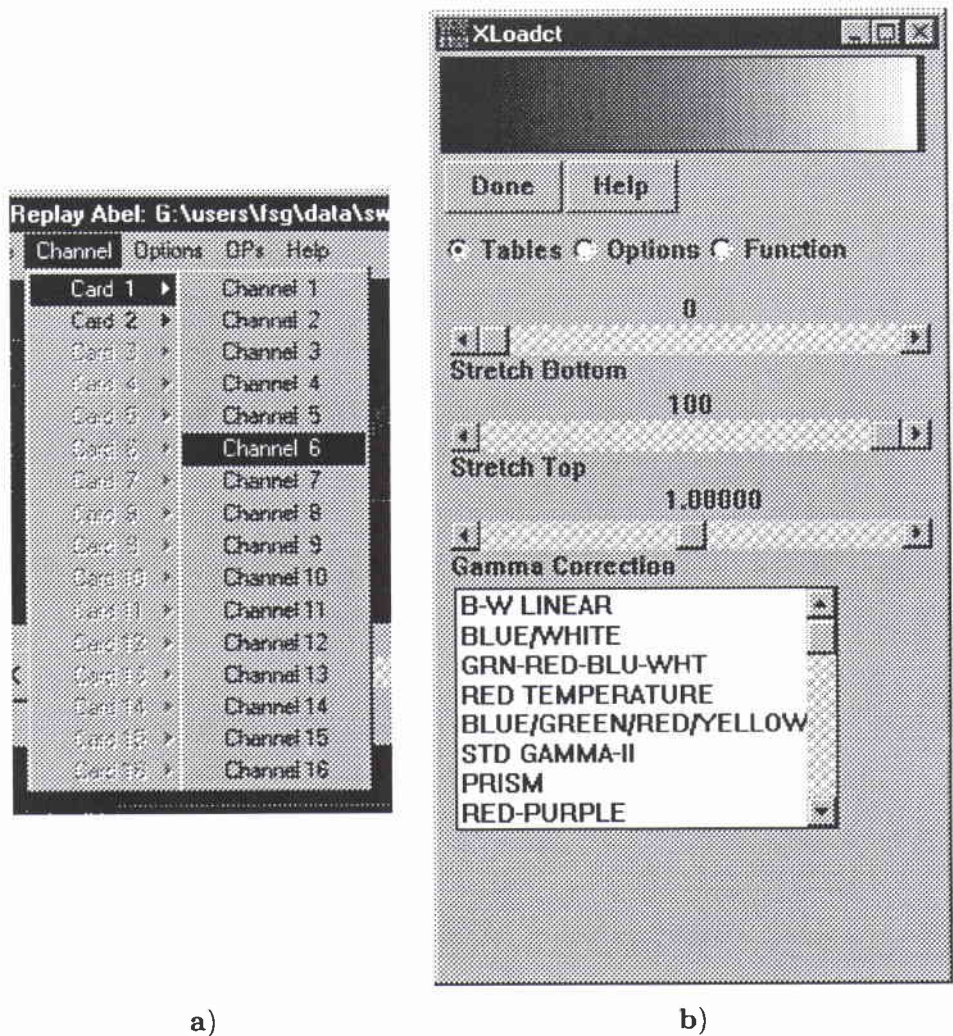


Figure 3: a) An example of how to select channel 6 card 1 pings; b) an example of the widget used to load the color table (Windows[®] environment).

Data information A number of information is available using the status bar. After having loaded a data file, it is possible to know which ping, channel, and card is displayed (see Fig. 1). Pushing on the relative button it is then possible to obtain the information memorized in the relative header (see Fig. 4.a, and b, and Fig. 5). The information is displayed in a widget which can be left open while the data file is scanned or changed. The information will be updated as a consequence of a change of ping, or channel number. The button under the status bar is used to give information to the user: some mathematical routines use it.

NATO UNCLASSIFIED

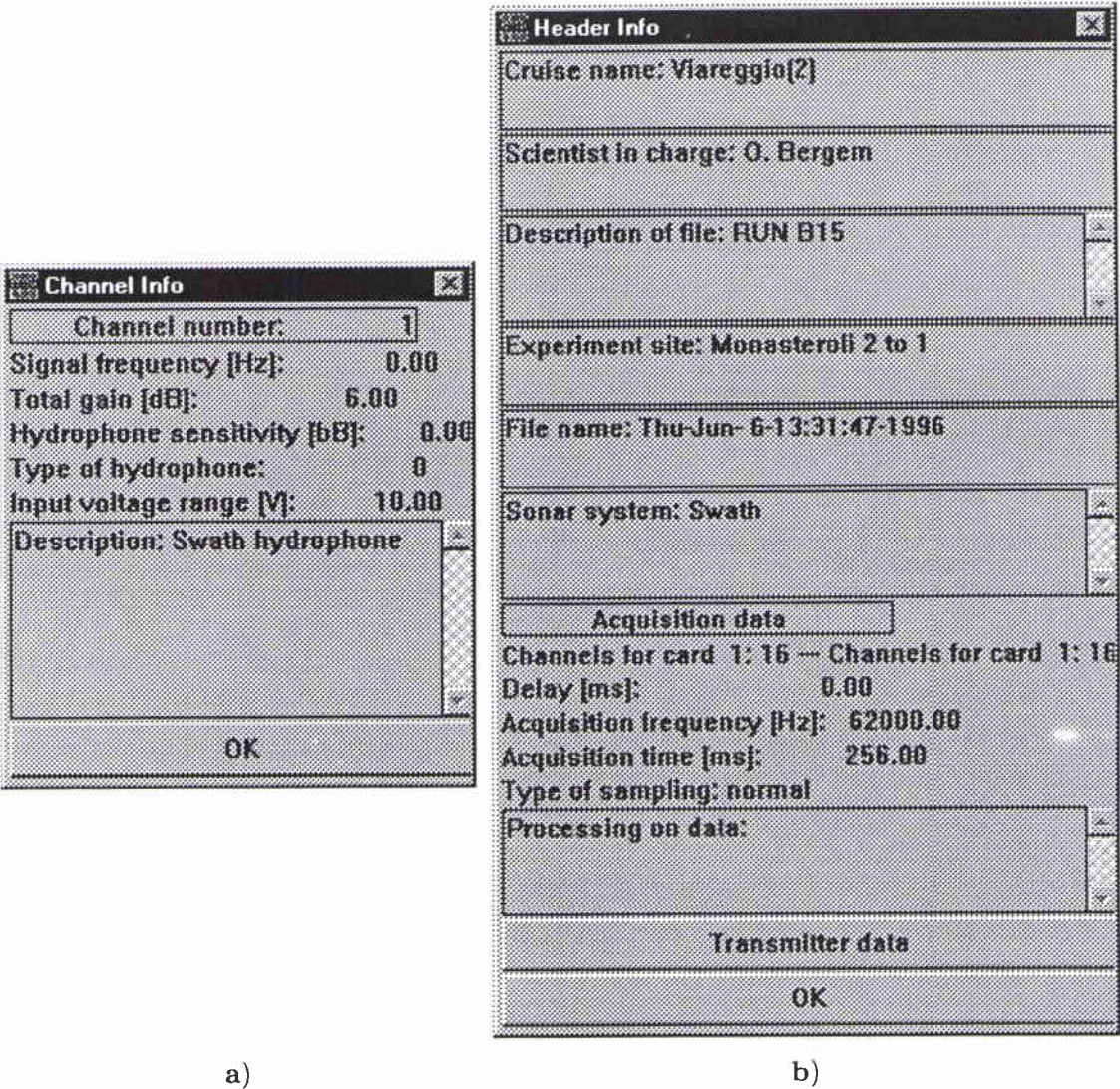


Figure 4: An example of the widgets showing the contents of the data file headers: the channel (a), and the main header headers (b).

Pick a point in the graph If a point in the graph is selected by clicking the mouse's center or right buttons, the point information is displayed in the **info** button of the status bar.

Data analysis The **OPs** item of the main window menu gives to the user a series of analysis operations (see Fig. 6). For a full description of these tools please consult the relative Subsection in the Section 14. The results of all the different operations are shown in the bottom drawing widget. Another important feature of these functions

NATO UNCLASSIFIED

SACLANTCEN M-127

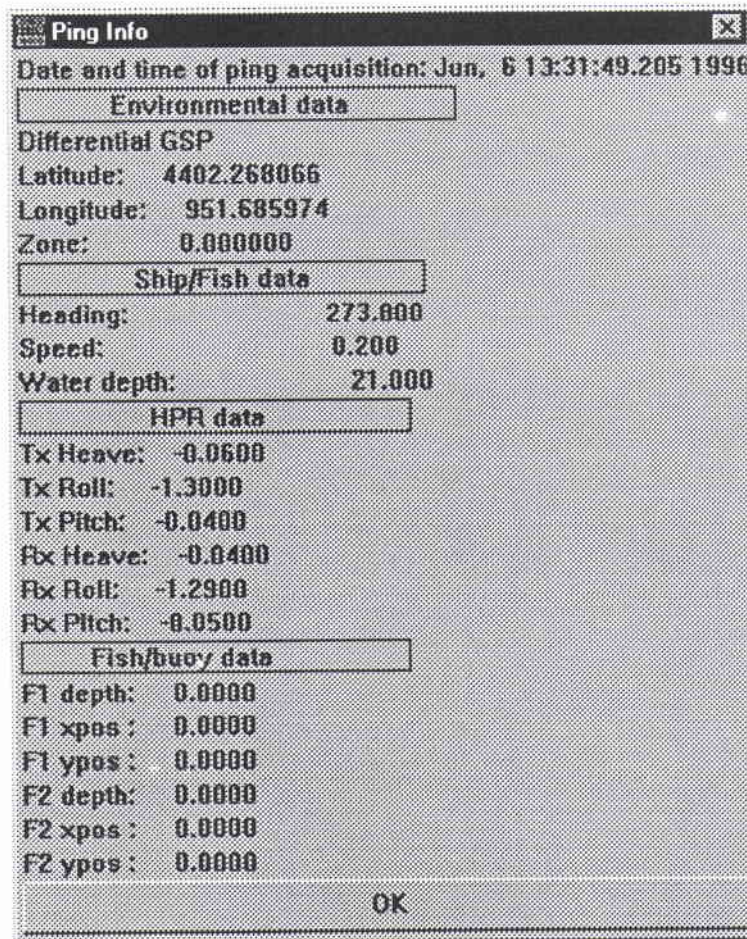
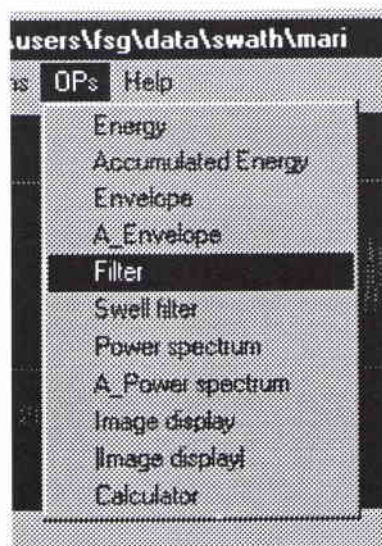


Figure 5: An example of the widgets showing the contents of the data file headers: the ping header.

is that they act on the selected part of the ping shown in the upper drawing window.

- **Energy:** This function calculates the total energy contained in the main window and shows the result in the **info** button.
- **Accumulated energy, Envelope, and Power spectrum:** These functions calculate the power (**Accumulated energy**), the power spectrum and the envelope of the signal.
- **Filter:** Clicking on this item displays an options widget (Fig. 8) with which it is possible to set up a band pass filter of appropriate order (see Subsection 14.16), together with a swell filter, and a high pass filter (zero mean). Clicking on the **Apply** button the filter is applied to the data. If the widget is

NATO UNCLASSIFIED

Figure 6: *The operations menu item.*

left open, the filter is applied to the data when other functions are applied to them (**Energy**, **Power spectrum**, *etc.*).

- **A-envelope**, **A-power spectrum**: These functions are similar to the **Envelope** and **Power spectrum** functions, except that they average their results on the selected ping range.
- **Swell filter**: This function acts on and attempt to allign a range of pings. A graph showing the delays which must be applied to align the pings is shown as a function of the ping number.
- **Image display** and **|Image display|**: These functions show, using false colour, an image built up allining a series of ping and placing in the abscissa-axis the time and in the ordinata-axis the ping number (see Figure fig:image®). The ping will vary in the ping range and the colour depth depends on the signal envelope (**Image display**) or on the signal absolute amplitude (**|Image display|**).
- **Calculator**: The last option will display a calculator which is able to perform all the operations just described, both on the ping or on the ping range. It is possible to perform other useful operations between more than one graph, such us: superimpose on the same coordinate system, cross-correlation and point to point difference.

NATO UNCLASSIFIED

SACLANTCEN M-127

2.2 *Function useable by the user*

This is an index of the functions useable out of **RAbel**:

load_io_file This function may be used to safely and interactively load a data file in **RAbel**.

load_io_ni This function may be used to safely but not interactively load a data file in **RAbel**.

dataset(), **datamat()**, and **dataping()** These functions are used to get the actual ordinate data vector (**dataset()**) and the actual matrix of ordinate data vectors selected using a range of ping greater than one (**datamat()**). The **dataping()** function obtains a matrix containing the full ping (all the channel data set).

xset() This function is used to obtain the actual abscissa data vector.

n_of_ping() This function is used to get the number of ping in the file.

get_data() This function is used to get the actual ping, channel, card and range numbers.

set_data This procedure is used to obtain the actual ping, channel, card and range numbers.

is_file_loaded() This function is used to ascertain if a file data is actually loaded.

data_file() This function is used to obtain the data file unit.

get_header(), **get_chan_head()**, **get_ping_head()** These functions return structure containing the respective Adam headers:

- the **adam_file_header**,
- the **adam_file_channel**, and
- the **adam_file_pinghead**.

getr_ping_head() This function returns an array of ping header structures containing the header of the actually selected pings.

get_version() This function returns the loaded data file version.

get_freq() This function is used to obtain the sampling frequency.

n_of_byte() This function is used to return the number of bytes used to memorize the sampled data.

NATO UNCLASSIFIED

n_of_chan() This function is used to obtain the total number of channel used during the experiment.

get_nochan() This function returns an array that contains the number of channel used on each acquisition card.

header_info(), **channel_info()**, **ping_info()**, and **transmitter_info()** These functions displays menus listing the information contained in the data file headers.

quit_rabel This procedure is used to safely quit RAbel.

help_rabel The help run the dvi viewer program to consult the electronic version of the manual.

about_rabel This procedure just shows an information widget with the release information.

reset_zoom This function resets the zooming.

set_title() This function sets the main window title.

redraw This procedure redraws the contents of the re-sizable widget inside the main widget.

convert This function is used to convert data type from hp, sun and PC to alpha format and vice-versa.

interval() This function returns the selected⁵ interval of the data.

pan This function moves the zooming windows left or right to permit an inquiry around the zooming window.

filter_wid This function is used to display the filter options widget and to apply the filter to the data.

print_graph This function is used to show the printing widget (which set up options and print, in a PostScript® file, the graphic widgets in the main window).

conv_head This function is used to convert data header type from hp, sun and PC to alpha format.

env This function is used to calculate the envelope of a vector.

mean This function is used to calculate the mean of a vector.

⁵By means of the zooming mechanism.

NATO UNCLASSIFIED

SACLANTCEN M-127

3

Main program: `rabel`

The script described in this section includes all the files containing the function definitions and runs a simple routine which calls the main window manager. Additional information on the functions contained in the included files is given in the relative sections of this manual.

3.1 `rabel`

Purpose This procedure starts **RAbel** (if not already running) or returns a warning message (if running).

Category main function

Usage

`rabel`

Common blocks This function uses the data stored in a common block that is initialized by the `io_rabel.com` file included in the `io_rabel.pro` file. See the last file for a description of the block.

Remarks The path where the **RAbel** files are must be in the `IDL_PATH`.

Description The routine checks for the presence of a **RAbel** widget:

- if present, **RAbel** stops writing a comment on the screen (multiple **RAbel** program executions are not allowed). The correct way to use **RAbel** is to restart it using the `.c` command.
- otherwise, the routine initializes the common block `DATA_FILE_Comm` with a NULL closed file and the path to search for data file (the actual IDL[®] directory). Finally the `cm_rabel` routine is called: it will open the main window widget managing the events and the graphic features.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```
<-----
; Include all the necessary functions
@io_rabel
@cm_rabel
@cm_rabel
@e_rabel
@g_rabel
@er_rabel
@sb_rabel
@op_rabel
@pl_rabel
@f_rabel
@ca_rabel
@type_con

PRO rabel

    IF XREGISTERED('MAIN_RABEL') THEN BEGIN
        PRINT, $
        'Rabel is already running. Please use ".c <ENTER>" to restart it!'
        RETURN
    ENDIF
; initialize the common block DATA_FILE_Comm with a NULL closed
; file and the path to search for data file with the actual IDL
; directory
    FREE_LUN, 20
    CLOSE, 20
    fileinfo=FSTAT(20)
    filepath='.'
    setfile, fileinfo, filepath
; run the main window creator
    cm_rabel

END
----->
```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

4

Data Management routines: `io_rabel`

This file contains the routines used to manage data file input/output. In particular, it opens and closes data files, changes channel, ping, card and range numbers, and obtains arrays containing abscissa and ordinates of the data.

4.1 `load_io_file`

Purpose This procedure is used to interactively load a file of **Adam** data: it checks for IO error and is quite safe.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
load_io_file
```

Common blocks This procedure uses the common blocks defined in the file: `io_rabel.com`. They are: `DATA_FILE_Comm`, `DATA_Comm`, and `POSITION_Comm`. These are the data blocks used by the Data Management Class of functions. Only this class can include these common blocks! The first common block contains the data file information. The second block contains all the structures used to store information about the **Adam** data being played. The third block is used to point to the actual ping (or ping range) selection. The next lines show the contents of `io_rabel.com`.

```
COMMON DATA_Comm, adam_file_id, adam_file_header, channels, $
    adam_file_tail, adam_file_pinghead, adam_size_ping, data, $
    id, header, tail, pinghead
COMMON DATA_FILE_Comm, FILEINFO, FILEPATH
COMMON POSITION_Comm, ping, card, channel, range
```

Remarks If the file selected is in an incorrect format, an error message is issued. If the tail of the file does not have the “magic number” **DADA** a warning is issued (see Subsection 16.3): it is still possible to load the file. If an error tag is included in the data file the error message is shown!

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

Description After the common blocks file inclusion the routine looks for a new file using the interactive **PICKFILE** function. Then it opens it using the **load_io_ni** function. The **PICKFILE** function pulls up the load file widget shown in Fig. 2.b.

```
; This declaration avoids compilation errors
FORWARD_FUNCTION main_id,is_file_loaded,conv_head,convert,get_filter, $
interval,get_version,n_of_chan,n_of_byte,get_nochan,create,get_freq,filter

PRO load_io_file
@io_rabel.com

; here we get the file
iid=main_id()
filename = PICKFILE(/READ,/MUST_EXIST,GROUP=iid, $
    PATH=FILEPATH, GET_PATH=FILEPATH)
load_io_ni, filename

END
```

4.2 load_io_ni

Purpose This procedure is used to not interactively load a file containing data stored using the **Adam** format: it checks for IO error and it quite safe.

Category input/output methods, function useable in the IDL[®] environment

Usage

load_io_file

Common blocks This function uses the common block included in the file the filename of which is **io_rabel.com**. This file is described in Subsection 4.1.

Description This routine must include the header file **io_rabel.hpr** where the **Adam** file structure information (common to different version of **Adam** files) is listed. **io_rabe1.hpr** and **io_rabe2.hpr** contains information which depends on the data file version. Depending on the **Adam** file version, the correct information is used. After the file inclusion the routine loads a new file, and closes the old data file (if it was open). In the event of an IO error the **CATCH** block catches the exception: this exception is dealt by opening a message box showing the error, then returning to the main window. After the file is opened (to be read) the file information (obtained using **FSTAT** IDL[®] function) is memorized in the common block **DATA_FILE_Comm** to be used to close the file or use it. The header data structures of the file are copied in the variables of the

NATO UNCLASSIFIED

SACLANTCEN M-127

Data_Comm common block. Before copying the information into the structures it is, if necessary, converted (swapping integer, long, and float bytes).

The tail is checked to verify that the tail identifier is present and whether flag is zero (success), greater than zero (warning) or less than zero (error). In case of warnings or errors a message box is displayed.

Subsequently, the number of pings acquired during the experiment is calculated using the equation given in Subsection 16.7. The routine associates the data vector to the first ping, first channel, first card stored in the input file (it uses the function `set_data`, described in this Section).

After these operations, the main window is redrawn using a series of graphic routines:

`reset_zoom`: resets the zoom factor (no zoom)

`set_title`: sets the main window title including the opened file name

`remake_m_menu`: makes two menus for card1 and card2 (if used)

`change_menu_bar`: activates the menu items that can be used when a file is loaded

`redraw`: redraws the main window with the updated value

Finally, the routine updates the status bar and the header information table (See Section 13)

```

PRO load_io_ni, filename
@io_rabel.com
@io_rabel.hpr
@io_rabe1.hpr
@io_rabe2.hpr

; this block catches exceptions
CATCH, Error_Status
    IF Error_Status NE 0 THEN BEGIN
        junk=['Error loading data file', !ERR_STRING]
        j=WIDGET_MESSAGE(junk, TITLE='Error!', /INFORMATION)
        RETURN
    ENDIF
; here it gets the new file
    IF STRLEN(filename) EQ 0 THEN RETURN
; it closes the old file and opens the new one
    IF FILEINFO.OPEN NE 0 THEN FREE_LUN, FILEINFO.UNIT
    OPENR, adam_file, filename, /GET_LUN
; and here it saves the data file information
    junk=FSTAT(adam_file)
    FILEINFO=junk

; here it associates the data file block to common structure

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

    id=ASSOC(adam_file, adamId)
; check if file was acquired by an HP
; if not, the program terminate
; ask to the author for a modification
    adam_file_id=id(0)
    IF (BYTE(adam_file_id.i_format) EQ 2) OR $
        (BYTE(adam_file_id.f_format) EQ 2) THEN BEGIN
            junk=['File not stored by an HP or compatible', $
                'RAbel does not work in this case!', $
                'Please ask to the author for more information.']
            j=WIDGET_MESSAGE(junk, TITLE='Fata Error', /ERROR)
            STOP
        ENDIF
    adam_file_id=conv_head(id(0))

CASE get_version() OF
    1: BEGIN
        adamSize=adamSize1
        adamHead=adamHead1
        adamChanHead=adamChanHead1
        adamPingHead=adamPingHead1
        adamTail=adamTail1
        END
    2: BEGIN
        adamSize=adamSize2
        adamHead=adamHead2
        adamChanHead=adamChanHead2
        adamPingHead=adamPingHead2
        adamTail=adamTail2
        END
    ENDCASE
    header=ASSOC(adam_file, adamHead, adamSize.Id)
    channels=ASSOC(adam_file, adamChanHead, adamSize.Id+ $
        adamSize.Head)
    tail=ASSOC(adam_file, adamTail, FILEINFO.SIZE-adamSize.Tail)
; before to copy the information into the structures it is,
; if necessary, converted
    adam_file_header=conv_head(header(0))
    adam_file_tail=conv_head(tail(0))
    nochan=get_nochan()
    chans=n_of_chan()

; check for tail 'check word'
    IF adam_file_tail.check NE FIX('DADA'X) THEN BEGIN
        junk=['Warning: this file has not 0xDADA in the tail.', $
            'It could be corrupted! Do you want to continue?']
        j=WIDGET_MESSAGE(junk, TITLE='Tail uncorrect!', /QUESTION)
        IF j NE 'Yes' THEN BEGIN
            CLOSE, adam_file

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

        junk=FSTAT(adam_file)
        FILEINFO=junk
        RETURN
    ENDIF
ENDIF

; check for data acquisition success
IF adam_file_tail.suc NE 0 THEN BEGIN
    IF adam_file_tail.suc GT 0 THEN BEGIN
        junk=['Warning: this file has a warning tag in its tail.', $
            'The warning comment is:',STRING(adam_file_tail.desc), $
            'Do you want to continue?']
        j=WIDGET_MESSAGE(junk, TITLE='Tail uncorrect!', $
            /QUESTION)
    ENDIF ELSE BEGIN
        junk=['Error: this file has an error tag in its tail.', $
            'The error comment is:',STRING(adam_file_tail.desc), $
            'Do you want to continue?']
        j=WIDGET_MESSAGE(junk, TITLE='Tail uncorrect!', $
            /QUESTION)
    ENDELSE
    IF j NE 'Yes' THEN BEGIN
        CLOSE, adam_file
        junk=FSTAT(adam_file)
        FILEINFO=junk
        RETURN
    ENDIF
ENDIF

; here the number of acquired pings is calculated
ping_length=adamSize.PingHead+adam_file_header.length* $
    chans*n_of_byte()
adam_size_ping=(FILEINFO.SIZE-adamSize.Tail-adamSize.Id- $
    adamSize.Head-chans*adamSize.ChanHead)/ping_length
pinghead=ASSOC(adam_file, adamPingHead, $
    adamSize.Id+adamSize.Head+chans*adamSize.ChanHead)
adam_file_pinghead=pinghead(0)
; here the program presume that at least a channel was acquired
ping=1
card=1
channel=1
range=1
set_data, ping, card, channel,range

; graphic routines are used to:
; reset zoom factor (no zoom)
; set the main window title including the just open file
; make the two menu for card1 and card2, if they are used
; activate the menu item that can be used when a file is loaded

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

; redraw the main window with the updated value
  reset_zoom
  set_title, 'Replay Abel: ' + filename
  FOR i=0,15 DO $
    remake_m_menu, 'Channel', i+1, 'Card '+STRING(i+1, $
      FORMAT='(i2)'), nochan(i)
; modify the status bar and the header information table
  change_menu_bar, 'Channel', 1
  change_menu_bar, 'Options', 1
  change_menu_bar, 'OPs', 1
  change_menu_bar, 'Print', 1
  set_slider
  remove_info
  redraw, /SB

```

END

4.3 get_data()

Purpose This function is used to return the ping, channel, card and range numbers.

Category input/output methods, function useable in the IDL® environment

Usage

```
v=get_data()
```

Output A vector which contains the four requested data items in the format [ping, card, channel, range].

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description This function returns a vector filled with the ping, card, channel, and range numbers stored in the common block `POSITION_Comm`.

```

FUNCTION get_data
@io_rabel.com

```

```
  RETURN, [ping, card, channel, range]
```

END

NATO UNCLASSIFIED

SACLANTCEN M-127

4.4 set_data

Purpose This function is used to set the ping, channel, and card numbers.

Category input/output methods, function useable in the IDL® environment

Usage

```
set_data, pingn, cardn, channeln, rangen
```

Arguments

pingn	the desired ping,
cardn	card,
channeln	channel, and
rangen	range numbers.

If they are set at zero their value is unchanged.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description The function accepts four parameters. If one of the parameters is zero, the value of the old parameter is left unchanged. The routine needs the information about data structure. Once the four parameters are set, the function stores the new (converted, if necessary) ping header in the common block. Then, the data structure is associated to the `data` variable (in the `DATA_Comm` common block) depending on the card number. The algorithm used to calculate the offset for the `ASSOC` function is tied to the structure of the data file.

The last thing the function does is to update the information included in the info widgets of ping and channel (if they are open).

```
PRO set_data, nping, ncard, nchannel, nrange
@io_rabel.com
@io_rabel.hpr
@io_rabe2.hpr
```

```
CASE get_version() OF
  1: BEGIN
    adamSize=adamSize1
    adamPingHead=adamPingHead1
  END
  2: BEGIN
    adamSize=adamSize2
    adamPingHead=adamPingHead2
  END
```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        ENDCASE
        chans=n_of_chan()
; if parameter is zero does not change the old value
        IF nping NE 0 THEN ping=nping
        IF ncard NE 0 THEN card=ncard
        IF nchannel NE 0 THEN channel=nchannel-1
        IF nrange NE 0 THEN range=nrange
; get the new ping header
        pinghead=ASSOC(FILEINFO.UNIT, adamPingHead, adamSize.Id+adamSize.Head+ $
            chans*adamSize.ChanHead+(ping-1)*(adamSize.PingHead+ $
            adam_file_header.length*chans*n_of_byte()))
        adam_file_pinghead=conv_head(pinghead(0))
; associate the new data structure depending on the card number
        chs=get_nochan()
        ch=FIX(TOTAL(chs(0:card-1))-chs(card-1))
        data=ASSOC(FILEINFO.UNIT, REPLICATE(1,chs(card-1), $
            adam_file_header.length),adamSize.Id+adamSize.Head+ $
            chans*adamSize.ChanHead+adamSize.PingHead+n_of_byte()*ch* $
            adam_file_header.length+(ping-1)*(adamSize.PingHead+ $
            adam_file_header.length*chans*n_of_byte()))
; reset information board for ping and channel
        IF nping NE 0 THEN change_ping_info
        IF nchannel NE -1 THEN change_channel_info

END

```

4.5 dataset()

Purpose This function is used to return the ordinate data vector.

Category input/output methods, function useable in the IDL® environment

Usage

```
v=dataset()
```

Output A vector which contains the ordinate data vector (it is pointed by the variables in the POSITION_Comm block).

Common blocks This function uses the common block included in the file the filename of which is io_rabel.com. This file is described in Subsection 4.1.

Description The function copies the matrix of data points in the tmp variable. Then it copies the channel header in the chan variable (depending from the card number and channel number) and uses the information in the header to re-scale the data to the original value. The original value V_o , in volt, is given

NATO UNCLASSIFIED

SACLANTCEN M-127

by:

$$V_o = \frac{10^{-G/20} R y}{S}$$

where G is the gain in dB (it is set in `chan.gain`), R is the input analog to digital voltage range (it is set in `chan.range`), y is the integer representing the data in the data matrix, and S is the integer scale factor for data, equal to the maximum absolute value of a sixteen bits signed integer.

The function returns a full array of data if the file is loaded; an error otherwise. To use this function out of `RAbel` it may be necessary to set a new ping number or a different channel. To perform this task it is possible to use the function `set_data` described later in this Section. The `get_data` function may be used to identify the actual ping, channel or card numbers.

FUNCTION dataset

@io_rabel.com

```

IF is_file_loaded() THEN BEGIN
    tmp=data(0)
    chs=get_nochan()
    ch=FIX(TOTAL(chs(0:card-1))-chs(card-1))
    chan=conv_head(channels(ch+channel))
    RETURN, TRANSPOSE((10^(-chan.gain/20.)*chan.range/ $
        (2^(n_of_byte()*8-1)-1))*convert(tmp(channel,*)))
ENDIF ELSE BEGIN
    junk=['Error: data file not loaded!']
    j=WIDGET_MESSAGE(junk, TITLE='File still unloaded!', $
        /INFORMATION)
    RETURN, 0
ENDELSE

```

END

4.6 xset()

Purpose This function is used to return the abscissa data vector.

Category input/output methods, function useable in the IDL® environment

Usage

```
v=xset()
```

Output An array which contains the abscissa data vector.

NATO UNCLASSIFIED

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description The function generates an array of floats with a content varying from 0.0 to the length of the data sample minus one, and increments of 1.0. Then the function adds the delay (that is given in sample number) and divides each element of the vector by the sampling frequency, obtaining the time at which each data sample was acquired.

```

FUNCTION xset
  @io_rabel.com

  IF is_file_loaded() THEN BEGIN
    tmp=FINDGEN(adam_file_header.length)
    RETURN, (tmp+adam_file_header.delay)/get_freq()
  ENDIF ELSE BEGIN
    junk=['Error: data file not loaded!']
    j=WIDGET_MESSAGE(junk, TITLE='File still unloaded!', $
      /INFORMATION)
    RETURN, 0
  ENDELSE
END

```

4.7 datamat()

Purpose This function is used to return the matrix of ordinate data vectors along a range of ping starting from the actual pings for a total of the **range** number of pings.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
v=datamat([/FILTERED,/INTER])
```

Output A matrix which contains the ordinate data vector (pointed by the common block `POSITION_Comm`).

Parameters

- FILTERED** If this parameter is set the data in the matrix are filtered before being returned.
- INTER** If this parameter is set only the selected part of the signal is returned in the matrix.

NATO UNCLASSIFIED

SACLANTCEN M-127

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description If the position has not changed, the function returns the data matrix memorized in the `MAT_TMP_Comm` block. Otherwise the function rebuilds the matrix: it creates a float matrix and then fills it using a `FOR` cycle. In the cycle it sets the ping number with the `set_ping()` function and obtains the data using the `dataset()` function. `set_ping()` is a function which sets the ping rapidly.

```

FUNCTION datamat, FILTERED=filtered, INTER=inter
@io_rabel.com
COMMON MAT_TMP_Comm, opi, oca, och, ora, matrix, change

  IF is_file_loaded() THEN BEGIN
    IF KEYWORD_SET(matrix) THEN $
      IF (opi EQ ping) AND (oca EQ card) AND (och EQ channel) AND $
        (ora EQ range) AND (NOT KEYWORD_SET(FILTERED)) AND change EQ 0 $
        AND (NOT KEYWORD_SET(INTER)) THEN RETURN, matrix
    opi=ping
    oca=card
    och=channel
    ora=range
    change=0
; create and fill the data matrix
    IF KEYWORD_SET(inter) THEN BEGIN
      v=interval()
      s=size(v)
      matrix=FLTARR(ora,s(s(0)),/NOZERO)
      FOR i=0,ora-1 DO BEGIN
        set_ping, opi+i
        matrix(i,*)=(dataset())(v)
      ENDFOR
      change=1
    ENDIF ELSE BEGIN
      matrix=FLTARR(ora,adam_file_header.length,/NOZERO)
      FOR i=0,ora-1 DO BEGIN
        set_ping, opi+i
        matrix(i,*)=dataset()
      ENDFOR
    ENDELSE
; reset the original ping number
    set_ping, opi
; filter the output if keyword FILTERED is on
    IF KEYWORD_SET(FILTERED) THEN $
      IF XREGISTERED('FILTER_WID') THEN BEGIN
        IF KEYWORD_SET(inter) THEN $

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        v=(xset()(interval())) $
    ELSE v=xset()
    filter, get_filter(), create([TRANPOSE(v), matrix], $
        ', ', ', ', ', ', 0, /MATRIX), /NOREDRA, /NOINTERVAL
    get_handle, inff, vetf
    vet=TRANPOSE(vetf)
    matrix=vet(1:range,*)
    change=1
    ENDIF
    RETURN, matrix
ENDIF ELSE BEGIN
; error!
    junk=['Error: data file not loaded!']
    j=WIDGET_MESSAGE(junk, TITLE='File still unloaded!', /INFORMATION)
    RETURN, 0
ENDELSE

END

```

4.8 dataping()

Purpose This function is used to return the matrix of ordinate data vectors, containing all the answers (channels) of a ping.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
v=dataping([/FILTERED] [, /INTERVAL] [, /ALLCARDS])
```

Output A matrix which contains the ordinate data vector (pointed by the variables in the `POSITION_Comm` block), containing all the data of a ping.

Parameters

FILTERED if this parameter is set, the data in the matrix are filtered before being returned

INTERVAL if this parameter is set, the matrix is built using only the selected interval;

ALLCARDS if this parameter is set, all the channels of all the used card are acquired.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description This function works like the `dataset()` function. The three options permit signal filtering, acquisition of the selected segment of signal only, and acquisition of all the channels (even from different card).

NATO UNCLASSIFIED

SACLANTCEN M-127

```

FUNCTION dataping, FILTERED=filtered, INTERVAL=inter, ALLCARDS=allcards
@io_rabel.com

IF is_file_loaded() THEN BEGIN
  chs=get_nochan()
IF KEYWORD_SET(ALLCARDS) THEN BEGIN
  tmpcard=1
  totchan=n_of_chan()
  tmp2=fltarr(totchan,adam_file_header.length,/NOZERO)
  tmp1=size(tmp2)
  tmpdata=get_data()
  WHILE chs(tmpcard-1) NE 0 DO BEGIN
    set_data, 0, tmpcard, 1, 0
    tmp=data(0)
    ch=FIX(TOTAL(chs(0:tmpcard-1))-chs(tmpcard-1))
    chan=conv_head(channels(ch+1))
    tmp2(ch:(ch+chs(tmpcard-1)-1),*)=(10^(-chan.gain/20.)*chan.range/ $
      (2^(n_of_byte()*8-1)-1))*convert(tmp(0:chs(tmpcard-1)-1,*))
    tmpcard=tmpcard+1
  END
  tmp=tmp2
  tmp2=0
  set_data, 0, tmpdata(1), tmpdata(2), 0
ENDIF ELSE BEGIN
  tmp=data(0)
  totchan=chs(card-1)
  ch=FIX(TOTAL(chs(0:card-1))-chs(card-1))
  chan=conv_head(channels(ch+channel))
  tmp1=size(tmp)
  tmp=(10^(-chan.gain/20.)*chan.range/(2^(n_of_byte()*8-1)-1))* $
    convert(tmp(0:chs(card-1)-1,*))
ENDELSE
; filter the output if keyword FILTERED is on
  junk={info_s, OP:0, data:[0, 0, 0, 0], interval:[0,0], $
    id:0L, name:'', XT:'', YT:'', freq:0.0, type:0, $
    baid:0L, btid:0L, ptid:0L}
  junk.data=FIX([ping,card,channel,totchan])
  junk.interval=[0,FIX(tmp1(tmp1(0))-1)]
  junk.type=1
  IF KEYWORD_SET(FILTERED) THEN $
    IF XREGISTERED('FILTER_WID') THEN BEGIN
      filter, get_filter(), create([TRANSPPOSE(xset()), tmp], $
        '','',',',0,/MATRIX, STRUCT=junk), /NOREDRAW
      get_handle, inff, vetf
      tmp=TRANSPPOSE(vetf)
      tmp=tmp(1:*,*)
    ENDIF
; select the output if the interval keyword is on

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        IF KEYWORD_SET(INTER) THEN tmp=tmp(*,interval())
        RETURN, tmp
    ENDIF ELSE BEGIN
        junk=['Error: data file not loaded!']
        j=WIDGET_MESSAGE(junk, TITLE='File still unloaded!', $
            /INFORMATION)
        RETURN, 0
    ENDELSE
END

```

4.9 n_of_ping()

Purpose This function is used to return the number of pings in the file.

Category input/output methods, function useable in the IDL® environment

Usage

```
v=n_of_ping()
```

Output An integer which gives the number of pings memorized in the file

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description This function returns the value of `adam_size_ping`! It is used to encapsulate the common data block.

```

FUNCTION n_of_ping
@io_rabel.com

    RETURN, adam_size_ping
END

```

4.10 set_ping

Purpose This function is used to set the ping for fast access to the data (used by `datamat()` function).

Category input/output methods, function useable in the IDL® environment

NATO UNCLASSIFIED

SACLANTCEN M-127

Usage`set_ping, pingn`*Arguments*`pingn` is the desired ping number (using the last value of card and channel).*Common blocks* This procedure uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.*Remarks* It is necessary to reset the ping number to the initial value at the end of the procedure to maintain the consistency of the program.*Description* The function accepts one parameter (the ping number) and sets the ping in a way similar to that of the `dataset()` function. After the use of this function, it is necessary to restore the ping number to the one which was active before the function use (otherwise the status bar information will not correspond to the ping).

```

PRO set_ping, nping
  @io_rabel.com
  @io_rabe1.hpr
  @io_rabe2.hpr

  CASE get_version() OF
    1: BEGIN
      adamSize=adamSize1
      adamPingHead=adamPingHead1
    END
    2: BEGIN
      adamSize=adamSize2
      adamPingHead=adamPingHead2
    END
  ENDCASE
  ping=nping
  chans=n_of_chan()
  chs=get_nochan()
  ch=FIX(TOTAL(chs(0:card-1))-chs(card-1))
  data=ASSOC(FILEINFO.UNIT, REPLICATE(1,chs(card-1), $
    adam_file_header.length),adamSize.Id+adamSize.Head+ $
    chans*adamSize.ChanHead+adamSize.PingHead+n_of_byte()*ch* $
    adam_file_header.length+(ping-1)*(adamSize.PingHead+ $
    adam_file_header.length*chans*n_of_byte()))

END

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

4.11 set_card_chan

Purpose This procedure is used to set the channel and card numbers given a string obtained from a menu event.

Category input/output methods, function not useable in the IDL® environment

Usage

set_card_chan, string

Arguments

string this is a string obtained from the menu event containing the card and the channel numbers.

Common blocks This procedure uses the common block included in the file the filename of which is io_rabel.com. This file is described in Subsection 4.1.

Description The procedure accepts a string obtained managing a menu event. It then removes the first part of the string (the word 'CHAN') and reads the two following integers (card and channel number). Then the function sets the new card and channel number and redraws the main windows maintaining the ping and range numbers.

```

PRO set_card_chan, s
  @io_rabel.com

  tmp=STRMID(s,4,STRLEN(s)-1)
  card=0 & channel=0
  READS, tmp, card, channel
  set_data, 0, card, channel, 0
  redraw, /SB

```

END

4.12 is_file_loaded()

Purpose This function is used from a number of routine to determine if data file is loaded.

Category input/output methods, function useable in the IDL® environment

Usage

q=is_file_loaded()

NATO UNCLASSIFIED

SACLANTCEN M-127

Output This function returns TRUE if an Adam format file is already loaded; FALSE otherwise.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description The function returns true if a data file is actually open. This information is stored into the `DATA_FILE_Comm` block.

```

FUNCTION is_file_loaded
@io_rabel.com

    RETURN, FILEINFO.OPEN NE 0

END

```

4.13 data_file()

Purpose This function is used to obtain the data file unit.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
unit=data_file
```

Output A long value which gives the value of the unit identifier of the data file

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Remarks This function should be used only if a file is already loaded.

Description This function returns the unit identifier associated with the data file.

```

FUNCTION data_file
@io_rabel.com

    RETURN, FILEINFO.UNIT

END

```

NATO UNCLASSIFIED

4.14 setfile

Purpose This procedure is used to set the COMMON DATA_FILE_Comm block and share the data file information and path.

Category input/output methods, function useable in the IDL® environment

Usage

setfile, fileinfo, path

Arguments

fileinfo this is the data file FSTAT structure obtained using the IDL® FSTAT function;

path this argument is a string containing the full data file path.

Common blocks This procedure uses the common block included in the file the filename of which is io_rabel.com. This file is described in Subsection 4.1.

Description This procedure sets the name and path of the data file.

```
PRO setfile, file, path
  @io_rabel.com
```

```
    FILEINFO=file
    FILEPATH=path
```

```
END
```

4.15 get_header()

Purpose This function is used to return the header of the data file.

Category input/output methods, function useable in the IDL® environment

Usage

v=get_header()

Output A structure which contains the header of the data file.

Common blocks This function uses the common block included in the file the filename of which is io_rabel.com. This file is described in Subsection 4.1.

Description It returns a structure containing the content of the adam_file_header variable.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

FUNCTION get_header
@io_rabel.com

    RETURN, adam_file_header

END

```

4.16 get_channel_head()

Purpose This function is used to return the channel header.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
v=get_channel_head()
```

Output A structure which contains the channel header.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description The function returns a structure containing the `adam_file_channel` header.

```

FUNCTION get_channel_head
@io_rabel.com

    chs=get_nochan()
    ch=FIX(TOTAL(chs(0:card-1))-chs(card-1))
    RETURN, conv_head(channels(ch+channel))

END

```

4.17 get_ping_head()

Purpose This function is used to return the header of the ping being read.

Category input/output methods, function useable in the IDL[®] environment

Usage

NATO UNCLASSIFIED

```
v=get_ping_head()
```

Output A structure which contains the ping header.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description This function returns a structure containing the ping header (a structure of `adam_file_pinghead` type).

```
FUNCTION get_ping_head
@io_rabel.com

    RETURN, adam_file_pinghead

END
```

4.18 getr_ping_head()

Purpose This function is used to return a vector containing the headers of the selected range of pings.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
v=getr_ping_head()
```

Output An array of structures which contains the ping headers of **range** pings, starting from the pointed ping.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description The function returns an array of structures containing the ping headers of the selected range of pings. It uses a **for** cycle to produce the array. To obtain an array of a single field of the ping header structure use the IDL[®] syntax `v.field`.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

FUNCTION getr_ping_head
@io_rabel.com
@io_rabe1.hpr
@io_rabe2.hpr

    dat=get_data()
    ran=dat(3)
; if range=1 the function return without delay the actual ping header
    if ran EQ 1 THEN $
        RETURN, adam_file_pinghead
; else construct and fill the array of structures
    CASE get_version() OF
        1: BEGIN
            adamSize=adamSize1
            adamPingHead=adamPingHead1
            END
        2: BEGIN
            adamSize=adamSize2
            adamPingHead=adamPingHead2
            END
    ENDCASE
    tmparr=replicate(adam_file_pinghead,ran)
    chans=n_of_chan()
    FOR i=1,ran-1 DO BEGIN
; get the new ping header
        ping=dat(0)+i
        pinghead=ASSOC(FILEINFO.UNIT, adamPingHead, $
            adamSize.Id+adamSize.Head+ $
            chans*adamSize.ChanHead+ $
            (ping-1)*(adamSize.PingHead+adam_file_header.length* $
            chans*n_of_byte()))
        tmparr(i)=conv_head(pinghead(0))
    END
    RETURN, tmparr

END

```

4.19 get_version()

Purpose This function is used to enquire about the Adam format version of the data file.

Category input/output methods, function useable in the IDL® environment

Usage

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```
v=get_version()
```

Output An integer denoting the Adam format version of the data file.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description It returns the data file version.

```
FUNCTION get_version
@io_rabel.com

  IF is_file_loaded() THEN $
  RETURN, adam_file_id.version AND 'FF'X $
  ELSE RETURN, 0
```

```
END
```

4.20 `get_freq()`

Purpose This function is used to return the sampling frequency.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
v=get_freq()
```

Output A float denoting the sampling frequency.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description This function returns the sampling frequency: it is used to encapsulate this data for different data file versions.

```
FUNCTION get_freq
@io_rabel.com

  IF is_file_loaded() THEN BEGIN
    CASE get_version() OF
      1: RETURN, adam_file_header.sfreq*1000.0
      2: RETURN, adam_file_header.sfreq
    ENDCASE
```

NATO UNCLASSIFIED

SACLANTCEN M-127

```
ENDIF ELSE RETURN, 0
```

```
END
```

4.21 n_of_byte()

Purpose This function is used to return the number of bytes used to memorize sampled data.

Category input/output methods, function useable in the IDL[®] environment

Usage

```
v=n_of_byte()
```

Output An integer denoting the number of bytes used to represent a datum.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description It returns the total number of bytes used to memorize data (for different data file versions).

```
FUNCTION n_of_byte
@io_rabel.com
```

```
IF is_file_loaded() THEN BEGIN
  CASE get_version() OF
    1: RETURN, 2
    2: RETURN, adam_file_header.noBits/8
  ENDCASE
ENDIF ELSE RETURN, 0
```

```
END
```

4.22 n_of_chan()

Purpose This function is used to return the total number of channels used during the experiment.

Category input/output methods, function useable in the IDL[®] environment

Usage

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

`v=n_of_chan()`

Output An integer denoting the number of channels used during the experiment.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description It returns the total number of channels used during the experiment (for different data file versions).

```

FUNCTION n_of_chan
@io_rabel.com

  IF is_file_loaded() THEN BEGIN
    CASE get_version() OF
      1: RETURN, adam_file_header.noChan
      2: RETURN, adam_file_header.totChan
    ENDCASE
  ENDIF ELSE RETURN, 0

END

```

4.23 `get_nochan()`

Purpose This function is used to return an array which contains the number of channels used on each acquisition card.

Category input/output methods, function useable in the IDL® environment

Usage

`v=get_nochan()`

Output A 16 integer array, each element of which is the number of channels used on the board.

Common blocks This function uses the common block included in the file the filename of which is `io_rabel.com`. This file is described in Subsection 4.1.

Description It builds and returns an array of integers the elements of which are the number of channels used on each board (a maximum of 16 cards is permitted starting from version 2 data file). It must deal with different versions of Adam format.

NATO UNCLASSIFIED

SACLANTCEN M-127

```
FUNCTION get_nochan  
  @io_rabel.com
```

```
  IF is_file_loaded() THEN BEGIN  
    CASE get_version() OF  
      1: BEGIN  
        tmp=intarr(16)  
        tmp=[adam_file_header.chan1,adam_file_header.chan2,$  
             0,0,0,0,0,0,0,0,0,0,0,0,0,0]  
        RETURN, tmp  
      END  
      2: RETURN, adam_file_header.noChan  
    ENDCASE  
  ENDIF ELSE RETURN, 0
```

```
END
```

NATO UNCLASSIFIED

5

Menu managing routines: `m_rabel`

These routines are used to create, manipulate (sensitize, de-sensitize, change sub-menus, etc.), and destroy the main menu bar.

The CW_PDMENU_S structure

This structure is similar to the `CW_PDMENU_S` structure defined in IDL®; one more flag is used to designate a help menu label which is displayed at the extreme right of the menu. The menu structure is then used by the `make_m_menu` and the `modify_m_menu` routines.



```
; this structures is described in the IDL User Manual
junk={CW_PDMENU_S, flags:0, name:''}
```

```
; this is the initial Rabel menu
MENU= [ $
  { CW_PDMENU_S,      1, 'File' },      $
    { CW_PDMENU_S,    0, 'Load' },      $
    { CW_PDMENU_S,    0, 'Print' },      $
    { CW_PDMENU_S,    0, 'Exit to IDL' }, $
    { CW_PDMENU_S,    2, 'Quit' },      $
  { CW_PDMENU_S,      1, 'Channel' },    $
    { CW_PDMENU_S,    1, 'Card 1' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 2' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 3' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 4' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 5' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 6' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 7' },    $
      { CW_PDMENU_S,  2, 'Channel 1' },  $
    { CW_PDMENU_S,    1, 'Card 8' },    $
```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 9' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 10' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 11' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 12' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 13' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 14' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Card 15' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 3, 'Card 16' }, $
        { CW_PDMENU_S, 2, 'Channel 1' }, $
    { CW_PDMENU_S, 1, 'Options' }, $
        { CW_PDMENU_S, 2, 'Load ColorTable' }, $
    { CW_PDMENU_S, 1, 'OPs' }, $
        { CW_PDMENU_S, 0, 'Energy' }, $
        { CW_PDMENU_S, 0, 'Accumulated Energy' }, $
        { CW_PDMENU_S, 0, 'Envelope' }, $
        { CW_PDMENU_S, 0, 'A_Envelope' }, $
        { CW_PDMENU_S, 0, 'Filter' }, $
        { CW_PDMENU_S, 0, 'Swell filter' }, $
        { CW_PDMENU_S, 0, 'Power spectrum' }, $
        { CW_PDMENU_S, 0, 'A_Power spectrum' }, $
        { CW_PDMENU_S, 0, 'Image display' }, $
        { CW_PDMENU_S, 0, '|Image display|' }, $
        { CW_PDMENU_S, 2, 'Calculator' }, $
    { CW_PDMENU_S, 9, 'Help' }, $
        { CW_PDMENU_S, 0, 'Help' }, $
        { CW_PDMENU_S, 2, 'About RAbel' } $
]

```

5.1 make_m_menu

Purpose This procedure is used to create the main window menu in such a way that it can be easily changed.

Category menu management methods, function not useable in the IDL[®] environment

Usage

```
make_m_menu, ch
```

NATO UNCLASSIFIED

Arguments

ch is a **string** used to identify the channel item (usually 'Channel').

Common blocks The only common block used is **MENU_Comm**, included in the file **m_rabel.com**. This is the block contains the main window menu information containing:

```
COMMON MENU_Comm, MAIN_MENUS_ID, MENU, menuid, cardid1, cardid2
```

Description This routine requires a slightly modified version of the **CW_PDMENU** function of IDL[®]. The only small difference in the routine is the possibility to use a fourth flag (4) to identify a help item for the menu (help item is placed on the right of the main window menu bar⁶). The structure of the menu is obtained from the file **m_rabel.hpr**. Before calling **CW_PDMER**, the procedure is to set the **U_VALUE** of the menu bar event in such a way that the events from the menu are sent to the correct event function by the **MAIN_RABEL_Event** routine⁷. The menu is created as a menu bar, returning the full name of the menu to allow identical names for sub-menus. The channel menu will be treated in a different way.

```
PRO make_m_menu
  @m_rabel.com
  @m_rabel.hpr
```

```
  WIDGET_CONTROL, MAIN_MENUS_ID, SET_UVALUE='MENU'
  M=CW_PDMER(MAIN_MENUS_ID,MENU,/MBAR,/RETURN_FULL_NAME, IDS=menuid)
```

```
END
```

5.2 remake_m_menu

Purpose This procedure is used to change the channel item of the main window menu.

Category menu management methods, function not useable in the IDL[®] environment

Usage

```
remake_m_menu, channel, cardn, card, n
```

⁶This option is not supported under Windows 3.1, Windows 95 and Windows NT.

⁷See Section 7.

NATO UNCLASSIFIED

SACLANTCEN M-127

Arguments

channel this is a string which identifies the channel item in the menu (usually 'Channel'⁸).

cardn this is the number of the card to be added (1 to 16).

card this is the string identifying the card item in the menu (usually 'Card 1', 'Card 2' etc.)⁹.

n this is the number of channels used in the card.

Common blocks This procedure also uses the **COMMON MENU_Comm** common block described in Subsection 5.1.

Description The procedure searches in the menu the index of the channel item; then uses this index to obtain the channel item ID. Using this identifier the procedure renders the channel item sensitive. Then the procedure looks for the card item index. If the number of the used channel is zero, the card item is desensitized. If the number is not zero the card item is sensitized, the old items are deleted and new items are reallocated. The identifiers of the allocated card item are memorized in a suitable structure for future deletion!

```

PRO remake_m_menu, channel, cardn, card, n
  Om_rabel.com

  i=0
; look for identifier index of card item
  WHILE MENU(i).name NE card DO i=i+1
; if number of channel used is zero desensitize it and return
  IF n EQ 0 THEN BEGIN
    WIDGET_CONTROL, menuid(i), SENSITIVE=0
    RETURN
  ; else
  ENDIF ELSE BEGIN
; get the correct structure for deletion
; sensitize the card item
    WIDGET_CONTROL, menuid(i), /SENSITIVE
; if it is the first time delete the originary menu id
    IF WIDGET_INFO(menuid(i+1), /VALID_ID) THEN $
      WIDGET_CONTROL, menuid(i+1), /DESTROY $
; else destroy all the item of the old card menu
  ELSE BEGIN
    IF cardn EQ 1 THEN cardid=cardid1 ELSE cardid=cardid2
    tmp_s=SIZE(cardid)
    FOR j=0, tmp_s(1)-1 DO $
      WIDGET_CONTROL, cardid(j), /DESTROY

```

⁸See the menu structure in the Subsection 5.

⁹See the menu structure.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        ENDELSE
; and now reconstruct them
        cardid=LONARR(n)
        FOR j=1,n DO $
            cardid(j-1)=WIDGET_BUTTON(menuid(i),VALUE='Channel '+ $
                STRING(j,FORMAT='(I2)'),'UVALUE='CARD'+ $
                STRING(cardn,FORMAT='(I2)')+ ' ' + STRING(j,FORMAT='(I2)') )
; memorizing the new item identifiers in the appropriate structure
            IF cardn EQ 1 THEN cardid1=cardid ELSE cardid2=cardid
        ENDELSE
END

```

5.3 save_menu_bar

Purpose This procedure stores the menu bar identifiers in a correct manner.

Category menu management methods, function not useable in the IDL[®] environment

Usage

```
save_menu_bar, id
```

Arguments

id this is the menu bar identifier as returned by WIDGET_BASE.

Common blocks This procedure also uses the COMMON MENU_Comm common block described in Subsection 5.1.

Description This routine stores the menu bar identifier in the appropriate common block.

```

PRO save_menu_bar, id
  @m_rabel.com

```

```
    MAIN_MENUS_ID=id
```

```
END
```

5.4 change_menu_bar

Purpose This procedure activates or deactivates an item of the menu bar.

NATO UNCLASSIFIED

SACLANTCEN M-127

Category menu management methods, function not useable in the IDL[®] environment

Usage

```
change_menu_bar, str, a
```

Arguments

str this is a menu bar item name (for the Channel menu it is usually 'Channel', for the Option menu it is usually 'Options', and for the Operation menu it is usually 'OPs').

a this is a parameter to activate (**a=1**) or deactivate (**a=0**) the menu item.

Common blocks This procedure also uses the **COMMON MENU_Comm** common block described in Subsection 5.1.

Description This routine enables or disables the appropriate menu item.

```
PRO change_menu_bar, str, a
  @m_rabel.com

  i=0
  WHILE MENU(i).name NE str DO i=i+1
  IF a EQ 0 THEN WIDGET_CONTROL, menuid(i), SENSITIVE=0 $
  ELSE WIDGET_CONTROL, menuid(i), SENSITIVE=1

END
```

NATO UNCLASSIFIED

6

Management of the main window: `cm_rabel`

This file contains the procedures which are used to create and display the main window and the procedures used to manage some widgets.

6.1 Main window dimensions: `cm_rabel.hpr`

This subsection describes a structure containing the optimum screen sizes used by the `main_w_resize` and the `main_w_create` routines (see after in the section). With the definition of the structure a global block containing the main window sizes is defined and initialized with optimum constants. Each system should use a different set of values.

```

;Define structures for RAbel

; This structure contains the dimensions used to build a nice
; main window
MAIN_W_SIZE = { SIZES, MIN_X_SIZE:0, MIN_Y_SIZE:0, MAX_X_SIZE:0, $
MAX_Y_SIZE:0, DRAW_W_SEP:0, S_B_POS:0, TOP_BAR:0, STATUS_BAR:0, $
INFO_BAR:0, BAR_SEP:0, MIN_R_MARGIN:0, MIN_L_MARGIN:0, $
MIN_T_MARGIN:0, MIN_B_MARGIN:0}

; this common block give to other routines access to this data
COMMON C_MAIN_WINDOW, W_SIZE
; x86 sizes
IF (!VERSION.ARCH EQ 'x86') THEN BEGIN
W_SIZE = { SIZES, $
    600,      $ ; MIN_X_SIZE minimum x size of the main window
    400,      $ ; MIN_Y_SIZE      "      y etc.
    1000,     $ ; MAX_X_SIZE maximum x etc.
    900,      $ ; MAX_Y_SIZE maximum y etc.
    65,       $ ; DRAW_W_SEP drawing window separation
    0,        $ ; S_B_POS slider bar position
    8,        $ ; TOP_BAR space between the bars and the
              $ ;      bottom drawing window
    22,       $ ; STATUS_BAR status bar height
    22,       $ ; INFO_BAR  info bar height
    4,        $ ; BAR_SEP   bars separation

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

    13,      $ ; MIN_R_MARGIN    "    right etc.
    5,      $ ; MIN_L_MARGIN    "    left margin of the etc.
    5,      $ ; MIN_T_MARGIN    "    top etc.
    60      $ ; MIN_B_MARGIN    "    bottom etc.
}
ENDIF ELSE BEGIN
; H3100 sizes
W_SIZE = { SIZES, $
    600,     $ ; MIN_X_SIZE minimum x size of the main window
    400,     $ ; MIN_Y_SIZE    "    y etc.
    1000,    $ ; MAX_X_SIZE maximum x etc.
    900,     $ ; MAX_Y_SIZE maximum y etc.
    65,      $ ; DRAW_W_SEP drawing window separation
    0,       $ ; S_B_POS slider bar position
    3,       $ ; TOP_BAR space between the bars and the
              $ ;                bottom drawing window
    22,      $ ; STATUS_BAR status bar height
    22,      $ ; INFO_BAR  info bar height
    3,       $ ; BAR_SEP   bars separation
    5,       $ ; MIN_R_MARGIN    "    right etc.
    5,       $ ; MIN_L_MARGIN    "    left margin of the etc.
    5,       $ ; MIN_T_MARGIN    "    top etc.
    3        $ ; MIN_B_MARGIN    "    bottom etc.
}
ENDELSE

```

6.2 main_w_create

Purpose This procedure is used to create the resizable widgets of the main window.

Category widget function, function not useable in the IDL[®] environment

Usage

```
main_w_create, wg
```

Arguments

wg this is the widget geometry descriptor of the main window as described in the IDL manual¹⁰

Common blocks The `cm_rabel.com` file is included in this routine. It contains the following common blocks definitions: `C_MAIN_WINDOW`, `DRAWU_Comm`, `DRAWD_Comm`, `MAIN_Comm`, `PING_SLIDER_Comm`, and `SHIFT_KEYS_Comm`. These are the common blocks containing: the widget identifiers and the drawing window identifiers of

¹⁰See IDL[®] Reference Guide, `WIDGET_INFO`, keyword `GEOMETRY`.

NATO UNCLASSIFIED

- 50 -

the resizable objects in the main window¹¹, the widget identifier of the main window¹², and the correct position of the various resizable objects inside the main window¹³.

The content of the `cm_rabel.com` file is:

```
COMMON DRAWU_Comm, DRAWUID
COMMON DRAWD_Comm, DRAWDID
COMMON MAIN_Comm, MAINID
COMMON PING_SLIDER_Comm, PING_SLIDERID
COMMON SHIFT_KEYS_Comm, SHIFT_KEY_LID, SHIFT_KEY_RID
```

Description At the beginning of the routine (after the blocks declarations) the dimensions and the positions of the resizable objects in the main window are defined. Sizes and positions take into account the main window dimensions (contained in the `wg` structure) and the correct position of the various resizable objects inside the main window (contained in the `W_SIZE` structure). The status bar is created using the function `make_status_bar` (see Subsection 13.1); `DRAWU` is the top drawing window, `DRAWD` is the bottom drawing window, and `PING_SLIDER` is the ping selector slider.

```
PRO main_w_create, wg
@cm_rabel.com
@cm_rabel.hpr

tmp_x_len=wg.XSIZE-W_SIZE.MIN_R_MARGIN-W_SIZE.MIN_L_MARGIN
tmp_y_len=(wg.YSIZE-W_SIZE.MIN_B_MARGIN-W_SIZE.MIN_T_MARGIN-$
W_SIZE.DRAW_W_SEP-W_SIZE.TOP_BAR-W_SIZE.STATUS_BAR-$
W_SIZE.INFO_BAR-W_SIZE.BAR_SEP)/2
; create the status bar
make_status_bar, wg
; create the top draw widget leaving enough margin around it
DRAWU = WIDGET_DRAW( MAINID, $
    BUTTON_EVENTS=1, $
    RETAIN=2, $
    UVALUE='DRAWU', $
    XSIZE=tmp_x_len, $
    YSIZE=tmp_y_len, $
    XOFFSET=W_SIZE.MIN_L_MARGIN, $
    YOFFSET=W_SIZE.MIN_T_MARGIN)
; create the slider used to choose the ping number
PING_SLIDER = WIDGET_SLIDER( MAINID, $
    MAXIMUM=200, $
```

¹¹`DRAWU_Comm`, `DRAWD_Comm`, `PING_SLIDER_Comm`, `SHIFT_KEYS_Comm`.

¹²`MAIN_Comm`.

¹³`C_MAIN_WINDOW`, a block that is defined in the `cm_rabel.hpr` file (see section 6.1).

NATO UNCLASSIFIED

SACLANTCEN M-127

```

    MINIMUM=1, $
    TITLE='Ping number', $
    UVALUE='PING_SLIDER', $
    VALUE=1, $
    SCROLL=1, $
    XSIZE=tmp_x_len-60, $
    XOFFSET=W_SIZE.MIN_L_MARGIN+30, $
    YOFFSET=tmp_y_len+W_SIZE.MIN_T_MARGIN+W_SIZE.S_B_POS)
; create the two shift keys
    SHIFT_KEY_L = WIDGET_BUTTON( MAINID, $
        VALUE='<<', $
        UVALUE='SHIFT_KEY_L', $
        XSIZE=25, $
        XOFFSET= W_SIZE.MIN_L_MARGIN, $
        YOFFSET=tmp_y_len+W_SIZE.MIN_T_MARGIN+13+W_SIZE.S_B_POS)

    SHIFT_KEY_R = WIDGET_BUTTON( MAINID, $
        VALUE='>>', $
        UVALUE='SHIFT_KEY_R', $
        XSIZE=25, $
        XOFFSET= W_SIZE.MIN_L_MARGIN+tmp_x_len-25, $
        YOFFSET=tmp_y_len+W_SIZE.MIN_T_MARGIN+13+W_SIZE.S_B_POS)
; create the bottom drawing widget
    DRAWD = WIDGET_DRAW( MAINID, $
        BUTTON_EVENTS=1, $
        RETAIN=2, $
        UVALUE='DRAWD', $
        XSIZE=tmp_x_len, $
        YSIZE=tmp_y_len, $
        XOFFSET=W_SIZE.MIN_L_MARGIN, $
        YOFFSET=tmp_y_len+W_SIZE.MIN_T_MARGIN+W_SIZE.DRAW_W_SEP)

```

Now that the three widgets are defined, their identifiers are copied in the associated block structures.

```

; Get drawing widgets index and ID
    DRAWUID=DRAWU
    DRAWDID=DRAWD
    PING_SLIDERID=PING_SLIDER
    SHIFT_KEY_LID=SHIFT_KEY_L
    SHIFT_KEY_RID=SHIFT_KEY_R

```

END

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

6.3 main_w_destroy

Purpose This procedure is used to destroy the resizable widgets of the main window in such a way that they can be redrawn with different sizes.

Category widget function, function not useable in the IDL[®] environment

Usage

```
main_w_destroy
```

Common blocks This procedure uses some of the common blocks defined in the `cm_rabel.com` files (see Subsection 6.2). It uses the common blocks that contain the identifiers of the resizable widgets in the main window.

Description This function destroys the resizable widgets in the main window with a call to `WIDGET_CONTROL` associated to the keyword `/DESTROY`. Then it destroys the status bar using the `destroy_status_bar` function¹⁴.

```
PRO main_w_destroy
```

```
  @cm_rabel.com
```

```
    WIDGET_CONTROL, PING_SLIDERID, /DESTROY
    WIDGET_CONTROL, DRAWUID, /DESTROY
    WIDGET_CONTROL, DRAWDID, /DESTROY
    WIDGET_CONTROL, SHIFT_KEY_LID, /DESTROY
    WIDGET_CONTROL, SHIFT_KEY_RID, /DESTROY
    destroy_status_bar
```

```
END
```

6.4 main_w_resize

Purpose This procedure is used to re-size the main window and the widgets inside it.

Category widget function, function not useable in the IDL[®] environment

Usage

```
main_w_resize, ev
```

Arguments

¹⁴See section 13

NATO UNCLASSIFIED

SACLANTCEN M-127

`ev` is the event descriptor of the main window (that is a `WIDGET_BASE`)¹⁵.

Common blocks This procedure uses only a common block defined in the file `cm_rabel.hpr` file (see Subsection 6.1): `C_MAIN_WINDOW`. This block contains information about the limit dimensions of the `RAbel` widget.

Description The procedure re-sizes the main window, then destroys the old re-sizeable objects, and redraws them with the correct dimensions.

If the dimension of `X` and `Y`SIZE are greater than a certain dimension, some workstations will crash. A test avoids main window dimensions which are excessive. If the main window dimensions are too small they are fixed to the dimensions memorized in the `W_SIZE` structure.

```

PRO main_w_resize, ev
COMMON C_MAIN_WINDOW, W_SIZE

; a test to avoid station crash!
  if(ev.X GT W_SIZE.MAX_X_SIZE) then ev.X=W_SIZE.MAX_X_SIZE
  if(ev.Y GT W_SIZE.MAX_Y_SIZE) then ev.Y=W_SIZE.MAX_Y_SIZE
  if(ev.X LT W_SIZE.MIN_X_SIZE) then ev.X=W_SIZE.MIN_X_SIZE
  if(ev.Y LT W_SIZE.MIN_Y_SIZE) then ev.Y=W_SIZE.MIN_Y_SIZE
; resize the main window
  WIDGET_CONTROL, ev.ID, XSIZE=ev.X, YSIZE=ev.Y
; destroy the old widget
  main_w_destroy
; and re-create them
  main_w_create, WIDGET_INFO(ev.ID, /GEOMETRY)
; then redraw their contents
  redraw, /DOWN

END

```

6.5 cm_rabel

Purpose This procedure is used to create and show the main window.

Category widget function, function not useable in the IDL[®] environment

Usage

```
cm_rabel, GROUP=gl
```

Parameters

¹⁵See the IDL[®] Reference Guide, `WIDGET_BASE`, keyword `TRACKING_EVENTS`.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

gl if this parameter is set, the RAbel widget is created using the widget identifier of another group leader (useful if this routine is called by another widget¹⁶)

Common blocks This procedure uses only a common block defined in the file `cm_rabel.com` (see Subsection 6.3): **MAIN_Comm**. This block contains the identifier of the main RAbel widget.

Description This routine defines and allocates the main window widget with a menu bar. The menu bar identifier is saved in the appropriate common block. Then the main window menu is created (using the `make_m_menu` function), and the re-sizeable widget in the main window is allocated (using the `main_w_create` function). At the end the widget is realized on the screen and the event control given to **XMANAGER**.

```

PRO cm_rabel, GROUP=Group
COMMON MAIN_Comm, MAIN_RABEL

  IF N_ELEMENTS(Group) EQ 0 THEN Group=0
; define the main window
  MAIN_RABEL = WIDGET_BASE(GROUP_LEADER=Group, $
    XSIZE=500,          $ ; normal main window xsize
    YSIZE=500,          $ ; normal main window ysize
    MAP=1,              $
    TITLE='Replay Abel', $ ; main window title
    UVALUE='MAIN_RABEL', $ ; event name
    /TLB_SIZE_EVENT,    $ ; request for resize event
    MBAR=PDM            $ ; this create a nice menu bar!
  )
; save the menu bar ID
  save_menu_bar, PDM
; create the main window menu
  make_m_menu
  change_menu_bar, 'Channel', 0
  change_menu_bar, 'Options', 0
  change_menu_bar, 'OPs', 0
  change_menu_bar, 'Print', 0
; create the resizeable widgets looking for the main window geometry
  main_w_create, WIDGET_INFO(MAIN_RABEL, /GEOMETRY)
; realize the main window
  WIDGET_CONTROL, MAIN_RABEL, /REALIZE
; manage the main window
  XMANAGER, 'MAIN_RABEL', MAIN_RABEL

END

```

¹⁶See IDL® Reference Guide, **WIDGET.BASE**, keyword **GROUP_LEADER**).

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

6.6 main_id()

Purpose This function is used to return the main window id.

Category widget function, function not useable in the IDL[®] environment

Usage

```
id=main_id()
```

Common blocks This function uses only a common block defined in the file `cm_rabel.com` (see Subsection 6.3): `MAIN_Comm`. This block contains the identifier of the main `RAbel` widget.

Description This function returns the main `RAbel` id. Only the Widget Manager class functions can use `MAIN_Comm`.

```

FUNCTION main_id
COMMON MAIN_Comm, MAIN_RABEL

```

```
    RETURN, MAIN_RABEL
```

```
END
```

6.7 quit_rabel

Purpose This procedure is used to quit `RAbel`.

Category widget function, function not useable in the IDL[®] environment

Usage

```
quit_rabel
```

Common blocks This function uses a common block defined in the `cm_rabel.com` file (see Subsection 6.3): `MAIN_Comm`. This block contains the identifier of the main `RAbel` widget.

Description This function (after a warning request) quits `RAbel` closing the opened data file.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```
PRO quit_rabel
COMMON MAIN_Comm, MAIN_RABEL

    junk=[' Are you sure you WANT', $
        '      to QUIT RABEL?']
    j=WIDGET_MESSAGE(junk, TITLE='Quit!', /QUESTION, /DEFAULT_NO)
    IF j EQ 'No' THEN RETURN
    IF is_file_loaded() THEN BEGIN
        CLOSE, data_file()
        FREE_LUN, data_file()
    ENDIF
    WIDGET_CONTROL, /DESTROY, MAIN_RABEL

END
```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

7

Event managing routine: e_rabel

This section contains the routines used to manage the events generated by the main window and main window menu-bar of **Rabel**. The functions which browse events are designed by an upper case descriptor followed by the word **Event**.

7.1 MAIN_RABEL_Event

Purpose This procedure is used to manage the main window events.

Category events manager routine, function not useable in the IDL[®] environment

Usage

MAIN_RABEL_Event, ev

Arguments

ev is the event structure describing the event¹⁷.

Description This routine calls a different procedure for each event generated.

```
FORWARD_FUNCTION get_freq, main_id, swell, create
```

```
PRO MAIN_RABEL_Event, Event
```

```
WIDGET_CONTROL, Event.Id, GET_UVALUE=Ev
```

```
CASE Ev OF
```

```
  'MAIN_RABEL': main_w_resize, Event ;Resize Event!
```

```
  'MENU': MENU_Event, Event ; Event for FILE_MENU
```

```
  'DRAWU': zoom_event, Event; Event of zooming!
```

```
  'PING_SLIDER': ping_slider_Event, Event
```

```
  'SHIFT_KEY_L': pan, -1
```

```
  'SHIFT_KEY_R': pan, 1
```

```
  'STATUS_BAR': STATUS_BAR_Event, Event
```

```
  'DRAWD': ZOOM_Event, Event; Event of zooming!
```

```
ENDCASE
```

¹⁷See the IDL[®] Reference Guide for each different event structures.

END

7.2 MENU_Event

Purpose This procedure is used to manage the main window menu events.

Category events manager routine, function not useable in the IDL[®] environment

Usage

MENU_Event, ev

Arguments

ev is the event structure describing the event¹⁸.

Description This routine calls a different procedure for each event generated. The Channel item is only treated in a different manner by the IF structure at the beginning of the procedure.

```
PRO MENU_Event, Event
```

```
    IF STRMID(Event.VALUE,0,4) EQ 'CARD' THEN set_card_chan, Event.VALUE $
    ELSE BEGIN
; time consuming cursor
        WIDGET_CONTROL, /HOURLASS
        CASE Event.VALUE OF
; load a new data file
            'File.Load': load_io_file
; destroy RAbel
            'File.Quit': quit_rabel
; print the graphic widgets of RAbel
            'File.Print': print_graph
; temporary exit to IDL
            'File.Exit to IDL': exit_to_idl
; Options menu
            'Options.Load ColorTable': BEGIN
                XLOADCT
            END
; ask for help
            'Help.Help': help_rabel
            'Help.About RAbel': about_rabel
; Operations call
```

¹⁸See the IDL[®] Reference Guide for each different event structures.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

'OPs.Accumulated Energy': a_energy, $
    create([TRANSPPOSE(xset()),$
        TRANSPPOSE(dataset())], '', '', '', 0)
'OPs.Energy': energy, create([TRANSPPOSE(xset()),$
    TRANSPPOSE(dataset())], '', '', '', 0)
'OPs.Envelope': envelope, create([TRANSPPOSE(xset()),$
    TRANSPPOSE(dataset())], '', '', '', 0)
'OPs.A_Envelope': a_envelope, create( $
    [TRANSPPOSE(xset()), datamat()], $
    '', '', '', 0,/MATRIX)
'OPs.Power spectrum': spectrum, create($
    [TRANSPPOSE(xset()),TRANSPPOSE(dataset())], $
    '', '', '', 0,FREQ=get_freq())
'OPs.A_Power spectrum': a_spectrum, create( $
    [TRANSPPOSE(xset()),datamat()], $
    '', '', '', 0,FREQ=get_freq(),/MATRIX)
'OPs.Image display': image, $
    create([TRANSPPOSE(xset()),$
        datamat()], '', '', '', 0,/MATRIX)
'OPs.|Image display|': image_abs, $
    create([TRANSPPOSE(xset()),$
        datamat()], '', '', '', 0,/MATRIX)
'OPs.Filter': filter_wid
'OPs.Swell filter': V=swell(TIME=1/get_freq(), $
    create([TRANSPPOSE(xset()),$
        datamat()], '', '', '', 0,/MATRIX))
'OPs.Calculator':make_calculator, GROUP=main_id()
ELSE:
ENDCASE
ENDELSE
END

```

NATO UNCLASSIFIED

8

Graphic routines: g_rabel

This section contains the routines used to manage the resizable widget of the main window. That means the two graph widgets and the ping slider.

8.1 zoom_event

Purpose This procedure is used to manage the zoom event (see Subsection 2.1).

Category graphic routine, function not useable in the IDL[®] environment

Usage

`zoom_event, ev`

Arguments

`ev` is the event structure describing the event¹⁹.

Common blocks This routine uses three common blocks: `ZOOM_Comm`, `ZOOM_local`, and `DRAWU_Comm`. The first is the common block which contains the zoom variables; the second is used by the zoom function only and the third contains the identifier of the upper drawing widget.

Description IDL[®] remembers the scaling of the last plotted graph only. As a consequence, it is necessary to redraw the plot before calling for a coordinate conversion!

This routine returns if no file was loaded. This function will be used with the zooming function to obtain information about the plotted graph. If the center or the left button of the mouse are clicked the pointed point coordinates are converted and written in the **info** button.

When the left button is pushed the pointed point is memorized (converted to graph coordinate) and the **z** flag set. If the left button is released the procedure first checks for an event which began out of the main window (probably an error) and then memorizes the point where the button was released, resetting the **z** flag.

The two coordinates are compared: if the horizontal component of the first

¹⁹See the IDL[®] Reference Guide for each different event structures.

NATO UNCLASSIFIED

SACLANTCEN M-127

zooming point is less than that of the second zoom point the points abscissa are memorized and the zooming flag is set. If greater, the zoom is reset. The zooming action is due to the redraw function that will plot the graph using the X_RANGE option set (only if zooming is set).

```

FORWARD_FUNCTION xset, dataset, is_file_loaded, main_id, get_data, $
is_operation, get_newdata, get_op, get_X_Title, get_Y_Title

PRO zoom_event, ev
COMMON DRAWU_Comm, DRAWUID
COMMON ZOOM_Comm, zooming, zoom_win_first, zoom_win_second
COMMON ZOOM_local, z, zwf, zws

; Do not do anything if file is not loaded
  IF NOT is_file_loaded() THEN RETURN
; if center or right button are clicked information are displayed
; about the point is picked
  IF((ev.PRESS NE 1) AND (ev.RELEASE NE 1)) THEN BEGIN
    IF (ev.TYPE EQ 1) THEN RETURN
; set the variable for convert_coord routine
    IF ev.ID EQ DRAWUID THEN redraw $
    ELSE redraw, /NOUP, /DOWN
    point=CONVERT_COORD(Ev.X, Ev.Y, /DEVICE, /TO_DATA)
    txt='(' + STRING(point(0), FORMAT='(F8.3)') + ', ' + $
      STRING(point(1), FORMAT='(F8.3)') + ')'
    change_info, txt
    IF ev.ID NE DRAWUID THEN redraw
    RETURN
  ENDIF
; if the left button is pushed
  tmp=get_header()
  IF (ev.TYPE EQ 0) AND (ev.ID EQ DRAWUID) THEN BEGIN
; set the variable for convert_coord routine
    zwf=CONVERT_COORD(Ev.X, Ev.Y, /DEVICE, /TO_DATA)
    IF zwf(0) LT tmp.delay/get_freq()*1000 THEN $
      zwf=tmp.delay/get_freq()*1000 $
    ELSE zwf=zwf(0)
    z=1
;and then it is released
  ENDIF ELSE BEGIN
    IF NOT KEYWORD_SET(z) THEN RETURN
    z=0
    zws=CONVERT_COORD(Ev.X, Ev.Y, /DEVICE, /TO_DATA)
    zws=zws(0)
; no zoom in case of right to left beginning
    IF zwf GE zws THEN zooming=0 $
    ELSE BEGIN

```

NATO UNCLASSIFIED

- 62 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        IF zws*get_freq()/1000 GT (tmp.length+tmp.delay) THEN $
            zoom_win_second=(tmp.length+tmp.delay)/get_freq()*1000 $
        ELSE zoom_win_second=FLOAT(zws)
            zoom_win_first=FLOAT(zwf)
            zooming=1
        ENDELSE
; redraw with zoom
    redraw
    ENDELSE

```

END

8.2 ping_slider_event

Purpose This function sets the data ping number with the event value of a ping_slider manipulation.

Category graphic routine, function not useable in the IDL[®] environment

Usage

```
ping_slider_event, ev
```

Arguments

ev is the event structure describing the event²⁰.

Description This routine does nothing if a file is not loaded; otherwise it sets the new ping value and redraws the graphic window.

```
PRO PING_SLIDER_Event, ev
```

```

    IF NOT is_file_loaded() THEN RETURN
    set_data, ev.VALUE, 0, 0, 0
    set_slider
    redraw, /SB

```

END

²⁰See the IDL[®] Reference Guide for each different event structures.

NATO UNCLASSIFIEDSACLANTCEN M-127**8.3 reset_zoom**

Purpose This procedure resets the zooming (used by functions that are not in the graphic class).

Category graphic routine, function useable in the IDL[®] environment

Usage

```
reset_zoom
```

Common blocks This procedure needs to have access to the ZOOM_Comm common block, which contains the zoom variables.

Description This routine just reset the zooming flag.

```

PRO reset_zoom
COMMON ZOOM_Comm, zooming, zoom_win_first, zoom_win_second

    zooming=0

END
  
```

8.4 interval()

Purpose This function gives the index range of the data samples to be shown.

Category graphic routine, function useable in the IDL[®] environment

Usage

```
ind=interval()
```

Common blocks This procedure needs to have access to the ZOOM_Comm common block, which contains the zoom variables.

Description This routine gives the index interval of the shown graph. It is used to perform mathematical operations on the data selected in the drawing screen.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

FUNCTION interval
COMMON ZOOM_Comm, zooming, zoom_win_first, zoom_win_second

  IF NOT is_file_loaded() THEN RETURN, 0
  tmp=SIZE(xset())
  mn=MIN(xset())
  mx=MAX(xset())
  k=tmp(1)/(mx-mn)
  IF zooming EQ 0 THEN RETURN, LINDGEN(tmp(1))
  firstp=ROUND(k*(zoom_win_first/1000.-mn))
  range=ROUND(k*(zoom_win_second-zoom_win_first)/1000.-1)
  RETURN, LINDGEN(range)+firstp

END

```

8.5 pan

Purpose This procedure moves the output window along the pings of a fixed quantity (2/3 of the screen length) to the right or left.

Category graphic routine, function useable in the IDL[®] environment

Usage

pan, lr

Arguments

lr this is the direction parameter: 1 means right-hand, -1 means left-hand.

Common blocks This procedure needs to have access to the ZOOM_Comm common block, which contains the zoom variables.

Description This routine calculates the upper drawing screen horizontal extension and, if lr EQ 1, sets two new values for the zooming parameters. Before setting the new zooming window the routine checks that the right margin is touched or overtaken, in which case, the zooming is clipped to the extreme right of the graph. The opposite is done if lr EQ -1.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

PRO pan, lr
COMMON ZOOM_Comm, zooming, zoom_win_first, zoom_win_second

  IF NOT is_file_loaded() THEN RETURN
  IF zooming EQ 0 THEN RETURN
  delta=(zoom_win_second-zoom_win_first)*2/3
  IF lr EQ 1 THEN BEGIN
    m=MAX(xset())*1000
    zoom_win_second=zoom_win_second+delta
    IF zoom_win_second GT m THEN zoom_win_second=m
    zoom_win_first=zoom_win_second-3*delta/2
  ENDIF ELSE BEGIN
    m=MIN(xset())*1000
    zoom_win_first=zoom_win_first-delta
    IF zoom_win_first LT m THEN zoom_win_first=m
    zoom_win_second=zoom_win_first+3*delta/2
  ENDELSE
  redraw

END

```

8.6 set_title

Purpose This function sets the main window title.

Category graphic routine, function useable in the IDL[®] environment

Usage

```
set_title, t
```

Arguments

t is the string containing the new title of the window.

Description This routine sets the new title. The main window identifier is obtained using the `main_id` function.

```

PRO set_title, t

  main_rabel=main_id()
  WIDGET_CONTROL, main_rabel, TLB_SET_TITLE=t

END

```

NATO UNCLASSIFIED

8.7 set_slider

Purpose This procedure sets the slider range of values.

Category graphic routine, function not useable in the IDL[®] environment

Usage

`set_slider`

Description This routine sets the new slider extension. The function takes into account the selected range of pings: if the ping plus the ping range is greater than the number of pings in the data file, then the ping is set in a congruent way. If the ping range is equal to the number of pings in the file the ping slider is set as not sensitive.

```

PRO set_slider
COMMON PING_SLIDER_Comm, PING_SLIDERID

; get the [ping, channel, card] vector and set the slider
tmp=get_data()
WIDGET_CONTROL, PING_SLIDERID, SET_VALUE=tmp(0), $
  SET_SLIDER_MAX=n_of_ping()-tmp(3)+1
IF n_of_ping() EQ tmp(3) THEN $
  WIDGET_CONTROL, PING_SLIDERID, SENSITIVE=0 $
ELSE WIDGET_CONTROL, PING_SLIDERID, SENSITIVE=1

END

```

8.8 redraw

Purpose This function redraws the contents of the resizable widgets inside the main widget.

Category graphic routine, function useable in the IDL[®] environment

Usage

`redraw [,/NOUP] [,/DOWN] [,/SB]`

Parameters

NOUP if this option is set, the upper graph will be not printed.
 DOWN if this option is set the lower graph is printed.
 SB if this option is set the status bar is rewritten.

NATO UNCLASSIFIED

SACLANTCEN M-127

Common blocks This routine uses four common blocks: DRAWU_Comm, DRAWD_Comm, ZOOM_Comm, and COLOR_Comm. The first two are the common blocks which contain the identifiers of the upper and lower drawing widget. The third contains the zoom variable and the fourth the colour to be used for multiple plots on the same graph.

Description If a data file is not loaded this routine does nothing. The function prints the status bar if necessary.

If the DOWN option is on, the handle containing information about the lower graphic window is loaded. The options 1 and 5 are the usual one; options 6 to 10 are used to overwrite a plot, while 11 to 15 are used to plot only the coordinate axis.

If the correct option is set the lower graph is plotted. The upper and lower windows are selected only if the output device is different to a PostScript[®] device.

```

PRO redraw, NOUP=noup, DOWN=down, SB=sb, ID=id
COMMON DRAWU_Comm, DRAWUID
COMMON DRAWD_Comm, DRAWDID
COMMON ZOOM_Comm, zooming, x1, x2
COMMON COLOR_Comm, COLORS, IND

; nothing happens if no file is loaded
  IF NOT is_file_loaded() THEN RETURN
; set new value in the status bar
  IF KEYWORD_SET(SB) THEN change_status_bar
; get the window index for drawing purpose
  WIDGET_CONTROL, DRAWUID, GET_VALUE=DRAWU_Id
  WIDGET_CONTROL, DRAWDID, GET_VALUE=DRAWD_Id
; plot the elaborated data if /DOWN key is on
; the command WSET is performed only for non PS
; graphic device
  IF KEYWORD_SET(DOWN) THEN BEGIN
; set the lower graphical widget as windows output
  IF !D.NAME NE 'PS' THEN WSET, DRAWD_Id
  get_handle, junk, w
  CASE junk.OP OF
    0:
      1: plot, 1000*w(*,0), 1000*w(*,1), $
        XTITLE=junk.XT, YTITLE=junk.YT, $
        XSTYLE=1
      2: plot, 1000*w(*,0), 1000*w(*,1), /XLOG, $
        XTITLE=junk.XT, YTITLE=junk.YT, $
        XSTYLE=1
      3: plot, 1000*w(*,0), w(*,1), $
        XTITLE=junk.XT, YTITLE=junk.YT, $

```

NATO UNCLASSIFIED

- 68 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```

XSTYLE=1
4: BEGIN
  Y=INDGEN(junk.data(3))+junk.data(0)
  X=w(0,*)
  Z=w(1:junk.data(3),*)
  PLOT, /NODATA, 1000*X, Y, $
  XTITLE=junk.XT, YTITLE=junk.YT, $
  XSTYLE=1, YSTYLE=1
  xybs=FIX(CONVERT_COORD(1000*MIN(X),MIN(Y),$
    /DATA,/TO_DEVICE))
  xyad=FIX(CONVERT_COORD(1000*MAX(X),MAX(Y),$
    /DATA,/TO_DEVICE))
  tmp=CONGRID(TRANPOSE(Z), $
    xyad(0)-xybs(0)-1, xyad(1)-xybs(1)-1)
  tvscl, tmp, xybs(0)+1, xybs(1)+1
END
5: plot, w(*,0), 1000*w(*,1), $
  XTITLE=junk.XT, YTITLE=junk.YT, $
  XSTYLE=1
6: BEGIN
  COLORS=COLORS-32
  IF COLORS LT 0 THEN COLORS=!P.COLOR
  oplot, 1000*w(*,0), 1000*w(*,1), COLOR=COLORS
END
7: BEGIN
  COLORS=COLORS-32
  IF COLORS LT 0 THEN COLORS=!P.COLOR
  oplot, 1000*w(*,0), 1000*w(*,1), /XLOG, COLOR=COLORS
END
8: BEGIN
  COLORS=COLORS-32
  IF COLORS LT 0 THEN COLORS=!P.COLOR
  oplot, 1000*w(*,0), w(*,1), COLOR=COLORS
END
10: BEGIN
  COLORS=COLORS-32
  IF COLORS LT 0 THEN COLORS=!P.COLOR
  oplot, w(*,0), 1000*w(*,1), COLOR=COLORS
END
11: plot, 1000*w(*,0), 1000*w(*,1), XSTYLE=1, /NODATA, /NOERASE
12: plot, 1000*w(*,0), 1000*w(*,1), XSTYLE=1, /NODATA, /NOERASE
13: plot, 1000*w(*,0), w(*,1), XSTYLE=1, /NODATA
15: plot, w(*,0), 1000*w(*,1), XSTYLE=1, /NODATA
ELSE:
ENDCASE
ENDIF
; set the upper graphical widget as windows output
  IF !D.NAME NE 'PS' THEN WSET, DRAWU_Id
; plot the original data

```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

```
IF (NOT KEYWORD_SET(NOUP)) THEN BEGIN
  xx=xset()
  i=interval()
  IF zooming EQ 0 THEN plot, 1000*xset(), 1000*dataset(), $
    XTITLE='time [ms]', YTITLE='Amplitude [mV]', XSTYLE=1 $
  ELSE plot, 1000*xx, 1000*dataset(),XRANGE=1000*[xx(min(i)), $
    xx(max(i))], XTITLE='time [ms]', YTITLE='Amplitude [mV]', XSTYLE=1
ENDIF
END
```

NATO UNCLASSIFIED

9

Print widget: pl_rabel

This section explains the routines used to print the two **RAbel** graphic windows. Clicking on the **File** menu item of the main **RAbel** window (Fig. 1) and then on the **Print** item the widget shown in Fig. 7 will be displayed.

9.1 print_graph

Purpose This procedure is used to show the printing widget.

Category user interface management method, function useable in the IDL[®] environment

Usage

`print_graph`

Common blocks This procedure uses the **PRINT_Comm** common block to save the information used to print file and to memorize the last printing configuration.

Description The procedure builds up a widget (not resizable border, no control button) into which are built a series of button groups used to choose the print setting (see Fig. 7). If the widget already exists it is popped up. The first two button groups are used to select the option of printing the upper or the lower graphic window (or both) and the color or type of PostScript[®] file. The encapsulated PostScript[®] file of IDL[®] cannot be printed if not inserted in an appropriate document (ie. L^AT_EX files). The second group of buttons (exclusive button) is used to set print font and sheet orientation. To print the graph, press the **Print** button, to close the widget press **Cancel**. Pressing **Print** saves the options. If it is the first time the widget is open the defaults are set, otherwise the data in the common block are used. The word “print” means that a file is created in which a Postscript[®] version of the graph is stored. To obtain a hard copy print the Postscript[®] file.

NATO UNCLASSIFIED

SACLANTCEN M-127

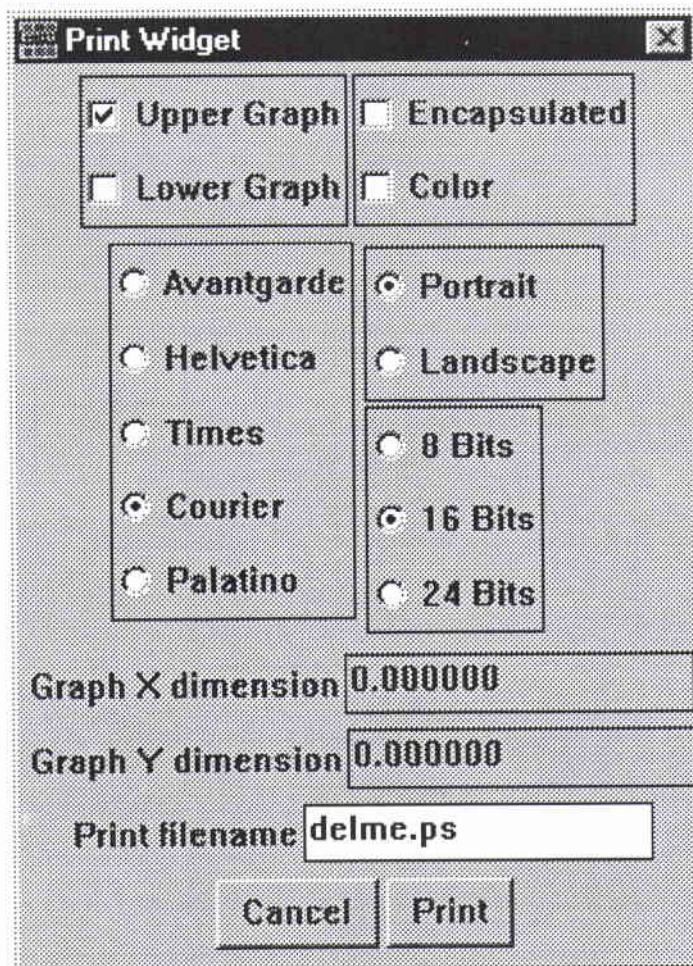


Figure 7: The appearance of the printing widget.

```

FORWARD_FUNCTION main_id

PRO print_graph
COMMON PRINT_Comm, P_base, P_bg1, P_bg2, P_bg3, P_bg4, P_bg5, $
P_bg6, P_field1, P_field2, P_field3, filename, bg1, bg2, bg3, $
bg4, bg5, bg6, xdim, ydim

    IF XREGISTERED('PRINT_GRAPH') THEN RETURN
    g=main_id()
; create the main widget
    IF !VERSION.ARCH EQ 'x86' THEN $
        P_base=WIDGET_BASE(GROUP=g, /COLUMN, TLB_FRAME_ATTR=1, $
            TITLE='Print Widget', /BASE_ALIGN_CENTER) $

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

ELSE P_base=WIDGET_BASE(GROUP=g, /COLUMN, TLB_FRAME_ATTR=11, $
    TITLE='Print Widget', /BASE_ALIGN_CENTER)
; create the two upper non exclusive button group
P_base2=WIDGET_BASE(P_base,/ROW)
P_bg1=CW_BGROUP(P_base2, /NONEXCLUSIVE, ['Upper Graph', $
    'Lower Graph'], /FRAME)
P_bg2=CW_BGROUP(P_base2, /NONEXCLUSIVE, ['Encapsulated', $
    'Color'], /FRAME)
; create the two exclusive button groups
P_base3=WIDGET_BASE(P_base,/ROW)
P_bg3=CW_BGROUP(P_base3, /EXCLUSIVE, ['Avantgarde', $
    'Helvetica', 'Times', 'Courier', 'Palatino'], /FRAME)
P_base4=WIDGET_BASE(P_base3,/column)
P_bg4=CW_BGROUP(P_base4, /EXCLUSIVE, ['Portrait', 'Landscape'], /FRAME)
P_bg6=CW_BGROUP(P_base4, /EXCLUSIVE, ['8 Bits', $
    '16 Bits', '24 Bits'], /FRAME)
P_field2=CW_FIELD(P_base,VALUE='',/FLOATING, TITLE='Graph X dimension')
P_field3=CW_FIELD(P_base,VALUE='',/FLOATING, TITLE='Graph Y dimension')
P_field1=CW_FIELD(P_base,VALUE='', STRING=1, TITLE='Print filename')
P_bg5=CW_BGROUP(P_base, ROW=1, ['Cancel', 'Print'], /RETURN_NAME)
; set default
IF (NOT KEYWORD_SET(filename)) THEN BEGIN
    filename='delme.ps'
    xdim=0
    ydim=0
    bg1=[1,0]
    bg2=[0,0]
    bg3=3
    bg4=0
    bg6=1
ENDIF
; set value in the window
WIDGET_CONTROL, P_field1, SET_VALUE=filename
WIDGET_CONTROL, P_field2, SET_VALUE=xdim
WIDGET_CONTROL, P_field3, SET_VALUE=ydim
WIDGET_CONTROL, P_bg1, SET_VALUE=bg1
WIDGET_CONTROL, P_bg2, SET_VALUE=bg2
WIDGET_CONTROL, P_bg3, SET_VALUE=bg3
WIDGET_CONTROL, P_bg4, SET_VALUE=bg4
WIDGET_CONTROL, P_bg6, SET_VALUE=bg6
; realize and manage the printing widget
WIDGET_CONTROL, P_base, /REALIZE
XMANAGER, 'PRINT_GRAPH', P_base

```

END

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

9.2 PRINT_GRAPH_Event

Purpose This procedure is used to manage the two printing widget events which are not automatically managed: **Print** and **Cancel**.

Category events manager routine, function not useable in the IDL[®] environment

Usage

PRINT_GRAPH_Event, ev

Arguments

ev is the event structure describing the event type²¹.

Common blocks This procedure uses the PRINT_Comm common block to save the information used to print file and to memorize the last printing configuration.

Description The only managed events are **Cancel** and **Print**: if the event is **Cancel**, the printing widget is destroyed; in the case of the **Print** event, the function memorizes the default and prints the graphic windows. To print set the device to 'PS', to set the option, and to call redraw with the correct options. The device is reset to the default.

```

PRO PRINT_GRAPH_Event, Event
COMMON PRINT_Comm, P_base, P_bg1, P_bg2, P_bg3, P_bg4, P_bg5, $
P_bg6, P_field1, P_field2, P_field3, filename, bg1, bg2, bg3, $
bg4, bg5, bg6, xdim, ydim

  IF Event.ID EQ P_bg5 THEN $
    CASE Event.VALUE OF
      'Cancel' : WIDGET_CONTROL, P_base, /DESTROY
      'Print' : BEGIN
        save=!D.NAME
        SET_PLOT, 'PS'
        WIDGET_CONTROL, P_bg2, GET_VALUE=bg2
        DEVICE, ENCAPSULATED=bg2(0)
        DEVICE, COLOR=bg2(1)
        WIDGET_CONTROL, P_bg3, GET_VALUE=bg3
        CASE bg3 OF
          0: DEVICE, /AVANTGARDE
          1: DEVICE, /HELVETICA
          2: DEVICE, /TIMES
          3: DEVICE, /COURIER
          4: DEVICE, /PALATINO
        ENDCASE
      
```

²¹ See the IDL[®] Reference Guide for each different event structures.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

WIDGET_CONTROL, P_bg4, GET_VALUE=bg4
CASE bg4 OF
  0: DEVICE, /PORTRAIT
  1: DEVICE, /LANDSCAPE
ENDCASE
WIDGET_CONTROL, P_bg6, GET_VALUE=bg6
CASE bg6 OF
  0: DEVICE, BITS_PER_PIXEL=8
  1: DEVICE, BITS_PER_PIXEL=16
  2: DEVICE, BITS_PER_PIXEL=24
ENDCASE
WIDGET_CONTROL, P_field1, GET_VALUE=filename
WIDGET_CONTROL, P_field2, GET_VALUE=XDIM
WIDGET_CONTROL, P_field3, GET_VALUE=YDIM
DEVICE, FILE=filename(0)
IF XDIM NE 0.0 THEN DEVICE, XSIZE=xdim
IF YDIM NE 0.0 THEN DEVICE, YSIZE=ydim
WIDGET_CONTROL, P_bg1, GET_VALUE=bg1
IF (bg1(0) EQ 1) AND (bg1(1) EQ 1) THEN !P.MULTI = [0,1,2]
IF bg1(0) EQ 1 THEN redraw
IF bg1(1) EQ 1 THEN redraw, /NOUP, /DOWN
DEVICE, /CLOSE
!P.MULTI = 0
SET_PLOT, save
IF bg1(1) EQ 1 THEN redraw
END
ENDCASE

```

END



NATO UNCLASSIFIED

SACLANTCEN M-127

10

Routines that implement events: `er_rabel`

This file contains the routines used to implement the event request generated by the `RAbel` program widgets.

10.1 `help_rabel`

Purpose The help is still not implemented. As a consequence this procedure shows an information widget.

Category general purpose methods function useable in the IDL[®] environment

Usage

`help_rabel`

Description This routine tries to start a dvi-viewer under **Windows** or **UNIX**. The name of these dvi-viewer must be:

`xdvi` under **UNIX**

`dviwin` under **Windows**

`dviwin2` under **Windows95**

If the dvi-viewer on your system has not such a name, an alias that point to its real name must be created. The executable of the program must be in the path. If your system was not in the preceding list it is necessary that the system is provided with a dvi-viewer the name of which must be included in the case structure.

The dvi-viewer is loaded using the dvi version of this manual.

PRO `help_rabel`

```
junk=[' Sorry, no Help file is available under VMS or Mac', $
      'Ask to Gaetano for a draft of the manual']
```

```
CASE !VERSION.OS OF
```

```
  'vms': GOTO, NOHELP
```

```
  'Win32': CMD = 'DVIWIN2 rabelman.dvi'
```

```
  'windows': CMD = 'DVIWIN rabelman.dvi'
```

NATO UNCLASSIFIED

- 76 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        'MacOS': GOTO, NOHELP
        ELSE: CMD = 'xdvi rabelman.dvi &'
    ENDCASE
    SPAWN, CMD
    RETURN
    NOHELP: j=WIDGET_MESSAGE(junk, TITLE='Sorry, no Help!', /INFORMATION)

END

```

10.2 about_rabel

Purpose This procedure shows an information widget with the release information.

Category general purpose methods, function useable in the IDL[®] environment

Usage

```
about_rabel
```

Description —No comment!—

PRO about_rabel

```

    junk=['Rabel: a program to replay Adam data (ver 2.2)', $
        '      NATO copyright 1996, NATO Unclassified', $
        '      Made by Gaetano Canepa, SACLANT Undersea Research Center, NATO']
    j=WIDGET_MESSAGE(junk, TITLE='About message', /INFORMATION)

END

```

10.3 exit_to_idl

Purpose This procedure returns the control to idl (Subsection 2.1).

Category general purpose methods, function not useable in the IDL[®] environment

Usage

```
exit_to_idl
```

NATO UNCLASSIFIED

SACLANTCEN M-127

Description The routine just stops the program! To return to the **XMANAGER** loop it is necessary to enter **.c** in IDL[®]. If a problem occurs during usage of IDL[®] the following commands should be entered in the same order:

- RETALL
- XMANAGER

and the control should return to **RAbel**.

```
PRO exit_to_idl
```

```
PRINT, '*****'
PRINT, '      RAbel stopped: TYPE <.c> to return to it'
PRINT, '*****'
STOP
```

```
END
```

NATO UNCLASSIFIED

11

Filtering widget: f_rabel

This section contains the routine used to display the filter options widget (Fig. 8). After the widget is activated and the options are set, it is possible to filter the data of the selected ping or ping range. Three routines are described here: the creator of the filter options widget, the event manager routine of this widget, and a function used to return the filter parameters.

When the filter widget is displayed, data filtering is performed on the data. Each time filtering is performed, the filter widget is popped up over all the other window as a remainder that a filter is in use.

11.1 filter_wid

Purpose This procedure is used to create the filter options widget.

Category user interface management method, function useable in the IDL[®] environment

Usage

`filter_wid`

Common blocks This procedure uses the **FILTER_Comm** common block. This block contains the widget identifiers used to change, obtain, and store the value of the widgets.

Remarks The match filter is still not implemented.

Description This procedure creates a main widget with some option buttons, and a series of slider widget to set the characteristic filter frequency. If the filter widget is shown for the first time, the setup values are applied to the widgets; otherwise the last operation values are used. Then, the base widget is realized and created.

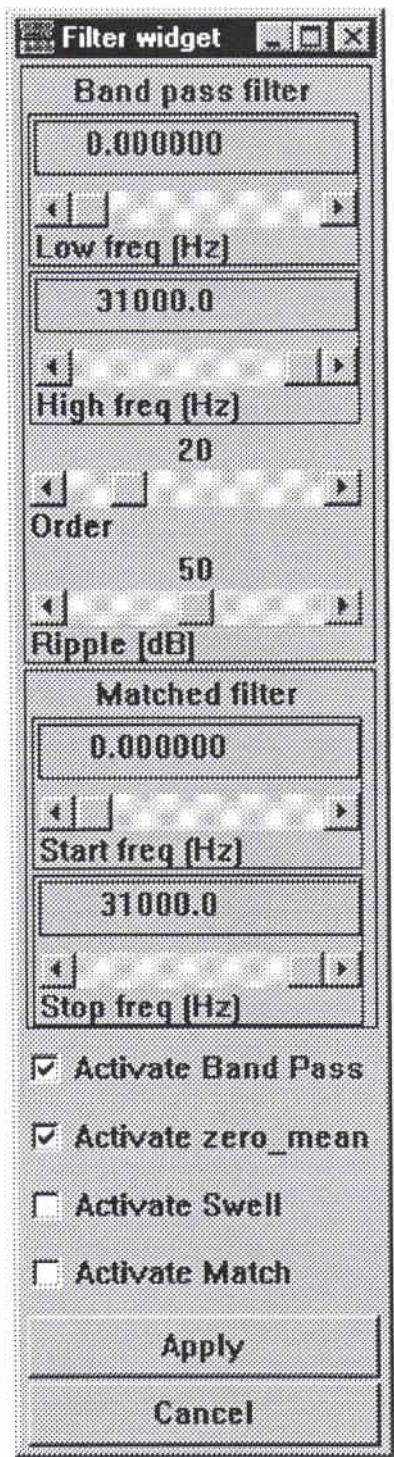


Figure 8: The appearance of the filter widget.

SACLANTCEN M-127

NATO UNCLASSIFIED

```

FORWARD_FUNCTION main_id, get_freq, new_handle

PRO filter_wid
COMMON FILTER_Comm, F_sl1, F_sl2, F_sl3, F_sl4, F_sl5, F_sl6, $
F_bg1, F_b1, F_b2, sl1, sl2, sl3, sl4, sl5, sl6, bg1

  IF XREGISTERED('FILTER_WID') THEN RETURN
  tfreq = get_freq()
  bas=main_id()
  F_base = WIDGET_BASE(COLUMN=1, TITLE='Filter widget', GROUP=bas)
  F_base1= WIDGET_BASE(F_base,COLUMN=1, /FRAME)
  F_label1 = WIDGET_LABEL(F_base1, VALUE='Band pass filter')
  F_sl1 = cw_fslir(F_base1, TITLE='Low freq (Hz)', $
    VALUE=0.0,MAXIMUM=tfreq/2.0,FRAME=1,/DRAG)
  F_sl2 = cw_fslir(F_base1, TITLE='High freq (Hz)', $
    VALUE=0.0,MAXIMUM=tfreq/2.0,FRAME=1,/DRAG)
  F_sl3 = WIDGET_SLIDER(F_base1, TITLE='Order', VALUE=0,MAXIMUM=100)
  F_sl4 = WIDGET_SLIDER(F_base1, TITLE='Ripple [dB]', VALUE=0, MAXIMUM=100)
  F_base2 = WIDGET_BASE(F_base,COLUMN=1,FRAME=1)
  F_label2= WIDGET_LABEL(F_base2, VALUE='Matched filter')
  F_sl5 = cw_fslir(F_base2, TITLE='Start freq (Hz)', $
    VALUE=0.0,MAXIMUM=tfreq/2.0,FRAME=1,/DRAG)
  F_sl6 = cw_fslir(F_base2, TITLE='Stop freq (Hz)', $
    VALUE=0.0,MAXIMUM=tfreq/2.0,FRAME=1,/DRAG)
  F_bg1 = CW_BGROU(P_F_base, /NONEXCLUSIVE, $
    ['Activate Band Pass', 'Activate zero_mean', $
    'Activate Swell', 'Activate Match'])
  F_b1 = WIDGET_BUTTON(F_base, VALUE='Apply')
  F_b2 = WIDGET_BUTTON(F_base, VALUE='Cancel')
; Set defaults
  IF (NOT KEYWORD_SET(sl1)) THEN BEGIN
    sl1=0
    sl2=tfreq/2.0
    sl3=20
    sl4=50
    sl5=0
    sl6=tfreq/2.0
    bg1=[1,1,0,0]
  ENDIF
; Set values
  WIDGET_CONTROL, F_bg1, SET_VALUE=bg1
  WIDGET_CONTROL, F_sl1, SET_VALUE=sl1
  WIDGET_CONTROL, F_sl2, SET_VALUE=sl2
  WIDGET_CONTROL, F_sl3, SET_VALUE=sl3
  WIDGET_CONTROL, F_sl4, SET_VALUE=sl4
  WIDGET_CONTROL, F_sl5, SET_VALUE=sl5
  WIDGET_CONTROL, F_sl6, SET_VALUE=sl6
; realize and manage the filter routine

```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

```
WIDGET_CONTROL, F_base, /REALIZE
XMANAGER, 'FILTER_WID', F_base
```

END

11.2 FILTER_WID_Event

Purpose This procedure is used to manage the event generated by the filter options widget.

Category events manager routine, function not useable in the IDL[®] environment

Usage

```
FILTER_WID_Event, ev
```

Arguments

ev is the event structure describing the event²².

Common blocks This procedure uses the **FILTER_Comm** common block. This block contains the widget identifiers used to change, obtain, and store the value of the widgets.

Description This procedure manages two of the possible events from the filter widget (the others are managed using default procedures). If the **Cancel** button is pressed (its identifier is **F_b2**) the options widget is destroyed; if the **Apply** button is pressed, the last option values are memorized and the filter is applied using the **filter** routine²³.

The behaviour of the function is different if the calculator widget is **on** or **off** (Section 15).

```
PRO FILTER_WID_Event, Event
COMMON FILTER_Comm, F_sl1, F_sl2, F_sl3, F_sl4, F_sl5, F_sl6, $
F_bg1, F_b1, F_b2, sl1, sl2, sl3, sl4, sl5, sl6, bg1

; Event handler for filter window
CASE event.Id OF
  F_b1: BEGIN
    WIDGET_CONTROL, F_sl1, GET_VALUE = sl1
    WIDGET_CONTROL, F_sl2, GET_VALUE = sl2
    WIDGET_CONTROL, F_sl3, GET_VALUE = sl3
```

²²See the IDL[®] Reference Guide for each different event structure.

²³See Section 14.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        WIDGET_CONTROL, F_s14, GET_VALUE = s14
        WIDGET_CONTROL, F_s15, GET_VALUE = s15
        WIDGET_CONTROL, F_s16, GET_VALUE = s16
        WIDGET_CONTROL, F_bg1, GET_VALUE = bg1
        f=get_freq()/2
junk={FILT_S, lf:0.0, hf:0.0, ord:0, rpl:0, $
      lfm:0.0, hfm:0.0, type:[0,0,0,0]}
;      IF XREGISTERED('MAIN_CALC') THEN BEGIN
;          filter, {FILT_S, s11/f, s12/f, s13, s14, s15/f, $
;              s16/f, bg1}, new_handle(), /CLEAN
;          set_calc
;      ENDIF ELSE $
          filter, {FILT_S, s11/f, s12/f, s13, s14, s15/f, s16/f, bg1}, $
          create([TRANPOSE(xset()), TRANPOSE(dataset())], ',',',',',',0)
;      END
      F_b2: WIDGET_CONTROL, event.top, /DESTROY
      ELSE:
      ENDCASE
END

```

11.3 get_filter()

Purpose This procedure is used to obtain the filter options.

Category general purpose routine, function useable in the IDL[®] environment

Usage

```
s=get_filter()
```

Output A **FILT_S** structure filled with the filter options, containing the following data:

lf and **hf** are the low-pass, and high-pass frequency of the band-pass filter with respect to the sampling frequency (they vary between 0 and 0.5). If **lf**=0, then the filter is low-pass; if **hp**=1/2 then the filter is high-pass.

ord and **rpl** are the order and the ripple factor of the FIR filter²⁴. **ord** is the number of samples minus one of the impulse responses of the FIR filter.

lfm and **hfm** are not used.

type is the type of filter implemented.

²⁴See the routine **DIGITAL_FILTER** in the IDL[®] Reference Guide.

NATO UNCLASSIFIED

SACLANTCEN M-127

Common blocks This procedure uses the **FILTER_Comm** common block. This block contains the widget identifiers used to change, obtains, and store the value of the widgets.

Description This procedure just the data in the **FILTER_Comm** common block and returns a structure containing information for the new filter.

```

FUNCTION get_filter
COMMON FILTER_Comm, F_sl1, F_sl2, F_sl3, F_sl4, F_sl5, F_sl6, $
F_bg1, F_b1, F_b2, sl1, sl2, sl3, sl4, sl5, sl6, bg1

  WIDGET_CONTROL, F_sl1, GET_VALUE = sl1
  WIDGET_CONTROL, F_sl2, GET_VALUE = sl2
  WIDGET_CONTROL, F_sl3, GET_VALUE = sl3
  WIDGET_CONTROL, F_sl4, GET_VALUE = sl4
  WIDGET_CONTROL, F_sl5, GET_VALUE = sl5
  WIDGET_CONTROL, F_sl6, GET_VALUE = sl6
  WIDGET_CONTROL, F_bg1, GET_VALUE = bg1
  f=get_freq()/2
  junk={FILT_S, lf:0.0, hf:0.0, ord:0, rpl:0, $
    lfm:0.0, hfm:0.0, type:[0,0,0,0]}
  RETURN, {FILT_S, sl1/f, sl2/f, sl3, sl4, sl5/f, sl6/f, bg1}

END

```

NATO UNCLASSIFIED

12

Data conversion routines: `type_con`

In this section some useful routines used to convert data between different hardware architectures are listed.

12.1 `convert()`

Purpose This function is used to convert data type from HP system to SUN, PC, and ALPHA systems format

Category general purpose routine, function useable in the IDL[®] environment

Usage

```
y=convert, val, CHANGE=val1
```

Output The converted value (array and matrix can be returned).

Parameters

CHANGE this is the parameter representing where the converted data should be copied; if not specified the data is simply returned by the function.

Arguments

val this is the data to be converted (arrays and matrixes are allowed).

Remarks Some of the options of this routine are not yet tested. It may happen, if data was acquired using a station other than an HP station, that **RAbel** quit when acquiring data. Ask the author for a modification of the program.

Description This function looks for the type of the data (integer, long or single and double floating) and then converts them using the appropriate IDL[®] function. The conversion is performed only for systems different to the HP (!VERSION.OS NE 'hp-ux').

NATO UNCLASSIFIED

SACLANTCEN M-127

```

FUNCTION convert, val2, CHANGE=val

  IF NOT KEYWORD_SET(val) THEN val = val2
  s=SIZE(val)
  k=s(0)+1
  CASE !VERSION.ARCH OF
    'alpha': BEGIN
      CASE s(k) OF
        '2': byteorder, val
        '3': byteorder,/LSWAP,val
        '4': byteorder,/XDRTOF,val
        '5': byteorder,/XDRTOD,val
      ELSE:
        ENDCASE
      END
    'x86': BEGIN
      CASE s(k) OF
        '2': byteorder, val
        '3': byteorder,/LSWAP,val
        '4': byteorder,/XDRTOF,val
        '5': byteorder,/XDRTOD,val
      ELSE:
        ENDCASE
      END
    ELSE:
      ENDCASE
  RETURN, val

END

```

12.2 conv_head()

Purpose This function is used to convert data header type from HP to SUN, PC, and ALPHA systems format.

Category general purpose routine, function useable in the IDL[®] environment

Usage

```
y=conv_head(str)
```

Output The converted header structure.

Arguments

str this is the data header structure to be converted.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

Description This function converts each tag of the header structure (if the variable needs to be converted). This is done only if the system is an alpha or a PC. The number of tags of the structure is detected using the IDL[®] functions N_TAGS.

```
FUNCTION conv_head, str
```

```
  IF (!VERSION.ARCH NE 'hp_ux') THEN $  
    FOR i=1,N_TAGS(str) DO str.(i-1)=convert(str.(i-1))  
  RETURN, str
```

```
END
```

NATO UNCLASSIFIED

SACLANTCEN M-127

13

Status bar routines: sb_rabel

This section contains the routines used to implement the status bar widget and to manage the status bar events.

13.1 make_status_bar

Purpose This procedure is used to create the status bar.

Category user interface management method, function not useable in the IDL[®] environment

Usage

`make_status_bar, wg`

Arguments

`wg` this is the widget geometry descriptor as described in the IDL manual²⁵.

Common blocks This procedure uses the `STATUS_BAR_Comm` common block. This block contains the variables where the status bar button identifiers are stored.

Description This routine creates the name of the buttons of the status bar adding the ping, channel, and card number to the label. Then it creates the buttons which are sensitized only when a file is loaded. The functions implementing the status bar are called by `STATUS_BAR_Event` using the `UVALUE` value.

```
FORWARD_FUNCTION info_id, get_header, get_channel_head, main_id, $
is_file_loaded, get_ping_head, get_nochan, get_freq, get_version, $
n_of_ping, get_data
```

```
PRO make_status_bar, wg
COMMON STATUS_BAR_Comm, IDS, INFO, in
@cm_rabel.hpr
```

²⁵ See IDL[®] Reference Guide, `WIDGET_INFO`, keyword `GEOMETRY`.

SACLANTCEN M-127

NATO UNCLASSIFIED

```

in=wg
MAINID=main_id()
tmp_x_len=wg.XSIZE-W_SIZE.MIN_R_MARGIN-W_SIZE.MIN_L_MARGIN
tmp_y_len=(wg.YSIZE-W_SIZE.MIN_B_MARGIN-W_SIZE.MIN_T_MARGIN-$
W_SIZE.DRAW_W_SEP-W_SIZE.TOP_BAR-W_SIZE.STATUS_BAR-$
W_SIZE.INFO_BAR-W_SIZE.BAR_SEP)/2
name=['Ping : ', 'Channel : ', 'Card : ', 'Header', 'Range : ']
IF is_file_loaded() THEN BEGIN
  data=get_data()
  name=name+[STRING(data(0), FORMAT='(I3)'), $
  STRING(data(2)+1, FORMAT='(I2)'), $
  STRING(data(1), FORMAT='(I2)'), ', ', $
  STRING(data(3), FORMAT='(I3)')]
  sensible=1
ENDIF ELSE BEGIN
  name=name+['-', '-', '-', '-', '-']
  sensible=0
ENDELSE
tmp=size(name)
ids=lonarr(tmp(1))
FOR i=0,tmp(1)-1 DO BEGIN
  IDS(i)=WIDGET_BUTTON(MAINID, $
  XSIZE=tmp_x_len/tmp(1), $
  YSIZE=W_SIZE.STATUS_BAR, $
  XOFFSET=W_SIZE.MIN_L_MARGIN+i*tmp_x_len/tmp(1), $
  YOFFSET=2*tmp_y_len+W_SIZE.MIN_T_MARGIN+ $
  W_SIZE.DRAW_W_SEP+W_SIZE.TOP_BAR, $
  VALUE=name(i), UVALUE='STATUS_BAR')
  WIDGET_CONTROL, ids(i), SENSITIVE=sensible
ENDFOR
INFO=WIDGET_BUTTON(MAINID, $
  XSIZE=tmp_x_len, $
  YSIZE=W_SIZE.INFO_BAR, $
  XOFFSET=W_SIZE.MIN_L_MARGIN, $
  YOFFSET=2*tmp_y_len+W_SIZE.MIN_T_MARGIN+ $
  W_SIZE.DRAW_W_SEP+ W_SIZE.STATUS_BAR+ $
  W_SIZE.TOP_BAR + W_SIZE.BAR_SEP, $
  VALUE='Info bar', UVALUE='STATUS_BAR')

```

END

13.2 set_range

Purpose This procedure is used to set the range value.

Category user interface management method, function useable in the IDI[®] environment

NATO UNCLASSIFIED

SACLANTCEN M-127

*Usage***set_range**

Common blocks This routine needs to define the SET_RANGE_Comm common block, where it stores the identifiers of the range modal widget.

Description This routine sets the new range value for ensemble operation. It uses a small modal widget. The range slider can vary only from 1 to the number of pings in the data file.

```

PRO set_range
COMMON SET_RANGE_Comm, b1, b2, slider, id

  IF !VERSION.ARCH EQ 'x86' THEN $
    id=WIDGET_BASE(TITLE='Set ping range', /COLUMN) $
  ELSE id=WIDGET_BASE(TITLE='Set ping range', /COLUMN, $
    TLB_FRAME_ATTR=11)
  slider=WIDGET_SLIDER(id, MAXIMUM=n_of_ping(), $
    MINIMUM=1, TITLE='Ping range', $
    VALUE=(get_data())(3), SCROLL=1)
  b1=WIDGET_BUTTON(id, VALUE='Set')
  b2=WIDGET_BUTTON(id, VALUE='Cancel')
  WIDGET_CONTROL, id, /REALIZE
  XMANAGER, 'SET_RANGE', id, /MODAL

END

```

13.3 SET_RANGE_Event

Purpose This procedure is used to manage the events generated when setting the range value.

Category user interface management method, function not useable in the IDL[®] environment

*Usage***SET_RANGE_Event, ev***Arguments*

ev is the event structure describing the event²⁶.

²⁶See the IDL[®] Reference Guide for each different event structures.

NATO UNCLASSIFIED

Common blocks This routine needs to define the `SET_RANGE_Comm` common block, where the identifiers of the range modal widget are stored.

Description This routine manages the events generated by the `set_range` widget: namely the OK and CANCEL events. The range value is obtained and set using the `set_data` function; then the widget is destroyed, the status bar is updated, and the ping slider adjusted if necessary.

```

PRO SET_RANGE_Event, ev
COMMON SET_RANGE_Comm, b1, b2, slider, id

CASE ev.ID OF
  b1: BEGIN
    WIDGET_CONTROL, slider, GET_VALUE=tmp
    d=get_data()
    p=d(0)
    IF (p+tmp) GT n_of_ping() THEN p=n_of_ping()-tmp+1
    set_data, p, 0, 0, tmp
    WIDGET_CONTROL, id, /DESTROY
    change_status_bar
    set_slider
  END
  b2: WIDGET_CONTROL, id, /DESTROY
ELSE:
ENDCASE

END

```

13.4 change_status_bar

Purpose This procedure is used to change (update) the status bar contents.

Category user interface management method, function useable in the IDL[®] environment

Usage

`change_status_bar`

Common blocks This procedure uses the `STATUS_BAR_Comm` common block. This block contains the variables where the status bar button identifiers are stored.

Description This routine destroys the old status bar and creates a new one.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

PRO change_status_bar
COMMON STATUS_BAR_Comm, IDS, INFO, WG
@cm_rabel.hpr

```

```

    destroy_status_bar
    make_status_bar, WG

```

```

END

```

13.5 destroy_status_bar

Purpose This procedure is used to destroy the status bar buttons.

Category user interface management method, function not useable in the IDL[®] environment

Usage

```

    destroy_status_bar

```

Common blocks This procedure uses the **STATUS_BAR_Comm** common block. This block contains the variables where the status bar button identifiers are stored.

Description This routine destroys all the status bar buttons; it obtains the identifiers from the common block.

```

PRO destroy_status_bar
COMMON STATUS_BAR_Comm, IDS, INFO, WG

    tmp=size(IDS)
    FOR i=0,tmp(1)-1 DO WIDGET_CONTROL, IDS(i), /DESTROY
    WIDGET_CONTROL, INFO, /DESTROY

```

```

END

```

13.6 STATUS_BAR_Event

Purpose This procedure is used to manage the status bar events.

Category events manager routine, function not useable in the IDL[®] environment

NATO UNCLASSIFIED

Usage

STATUS_BAR_Event, ev

Arguments

ev is the event structure describing the event²⁷.

Description This routine calls a different procedure for each event. It uses only the first four letters of the command to decide what to do. In the UVALUE of the OK button is memorized the base widget id, in such a way that it is possible to destroy it.

```

PRO STATUS_BAR_Event, Event

  WIDGET_CONTROL, EVENT.id, GET_VALUE=Ev
  CASE STRMID(Ev,0,4) OF
    'Head': header_info
    'Chan': channel_info
    'Ping': ping_info
    'OK': BEGIN
      WIDGET_CONTROL, EVENT.id, GET_UVALUE=Uv
      WIDGET_CONTROL, Uv, /DESTROY
    END
    'Tran': transmitter_info
    'Rang': set_range
  ELSE:
  ENDCASE

END

```

13.7 change_info

Purpose This procedure is used to write a string in the info button of the status bar.

Category widget function, function useable in the IDL[®] environment

Usage

change_info, str

Arguments

str this is the string to be displayed on the button.

²⁷See the IDL[®] Reference Guide for each different event structure.

NATO UNCLASSIFIED

SACLANTCEN M-127

Common blocks This procedure uses the `STATUS_BAR_Comm` common block. This block contains the variables where the status bar button identifiers are stored.

Description This function sets the value of the info button.

```

PRO change_info, txt
COMMON STATUS_BAR_Comm, IDS, INFO, WG
@cm_rabel.hpr

MAINID=main_id()
tmp_x_len=wg.XSIZE-W_SIZE.MIN_R_MARGIN-W_SIZE.MIN_L_MARGIN
tmp_y_len=(wg.YSIZE-W_SIZE.MIN_B_MARGIN-W_SIZE.MIN_T_MARGIN-$
    W_SIZE.DRAW_W_SEP-W_SIZE.TOP_BAR-W_SIZE.STATUS_BAR-$
    W_SIZE.INFO_BAR-W_SIZE.BAR_SEP)/2
WIDGET_CONTROL, INFO, /DESTROY
INFO=WIDGET_BUTTON(MAINID, $
    XSIZE=tmp_x_len, $
    YSIZE=W_SIZE.INFO_BAR, $
    XOFFSET=W_SIZE.MIN_L_MARGIN, $
    YOFFSET=2*tmp_y_len+W_SIZE.MIN_T_MARGIN+ $
    W_SIZE.DRAW_W_SEP+ W_SIZE.STATUS_BAR+ $
    W_SIZE.TOP_BAR + W_SIZE.BAR_SEP, $
    VALUE=txt, UVALUE='STATUS_BAR')

END

```

13.8 header_info

Purpose Show a widget containing the information in the file header.

Category widget function, function useable in the IDL[®] environment

Usage

`header_info`

Common blocks This procedure uses the `INFO_Comm` common block to store information about the created widget.

Description The first instruction pops up the header info widget (if it is already active); otherwise the routine begins to create it. It needs the main window identifiers to destroy the widget if the main window is closed. The base widget cannot be resized and cannot be closed (except using the OK button or if the PC version of `RAbel` is used). The identifiers of every widget used to display information, are memorized in the `INFO_Comm` common block, in the

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

array hei. After the routine has create all of the buttons or text widgets in the base widget, the base widget is realized and the control of the event is assigned using the `XMANAGER` function. The widget can be destroyed by pushing the OK button. Another button is used to display the transmitter information widget (see Subsection 13.10).

The header structure is obtained using the `get_header` function (see Section 4). In the next subsection the channel and the ping headers are also obtained using two other functions: `get_channel_head` and `get_ping_head`. The information of the header is displayed using standard widgets.

```

PRO header_info
COMMON INFO_Comm, pii, chi, tri, hei

  IF XREGISTERED('header_info') THEN RETURN
  month=['','Jan','Feb','Mar','Apr','May','Jun','Jul','Aug', $
    'Sep','Oct','Nov','Dec']
  id=main_id()
  adam_file_header=get_header()
  hei=lonarr(22)
  IF !VERSION.ARCH EQ 'x86' THEN $
    head_base=WIDGET_BASE(TITLE='Header Info', /COLUMN, $
      GROUP=id,TLB_FRAME_ATTR=1) $
  ELSE $
    head_base=WIDGET_BASE(TITLE='Header Info', /COLUMN, $
      GROUP=id,TLB_FRAME_ATTR=11)
  hei(0)=head_base
  hei(1)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
    VALUE='Cruise name: ' + $
    STRING(adam_file_header.cname), XSIZE=40, YSIZE=2)
  hei(2)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
    VALUE='Scientist in charge: ' + $
    STRING(adam_file_header.sic), XSIZE=40, YSIZE=2)
  hei(3)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
    VALUE='Description of file: ' + $
    STRING(adam_file_header.fdesc), XSIZE=40, YSIZE=2, /SCROLL)
  hei(4)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
    VALUE='Experiment site: ' + $
    STRING(adam_file_header.site), XSIZE=40, YSIZE=2)
  hei(5)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
    VALUE='File name: ' + $
    STRING(adam_file_header.fname), XSIZE=40, YSIZE=2)
  hei(6)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
    VALUE='Sonar system: ' + $
    STRING(adam_file_header.pdesc), XSIZE=40, YSIZE=2, /SCROLL)
  hei(7)=WIDGET_LABEL(head_base, /ALIGN_LEFT, /FRAME, $
    VALUE='          Acquisition data          ')
  nochan=get_nochan()

```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

```

FOR i=0,15,2 DO BEGIN
  IF nochan(i) GT 0 THEN BEGIN
    txt='Channels for card '+ STRING(i+1,$
    FORMAT='(I2)') +': ' + $
    STRING(nochan(i),FORMAT='(I2)')
    IF nochan(i+1) GT 0 THEN $
      txt=txt+' --- Channels for card '+ STRING(i+1,$
      FORMAT='(I2)') +': ' + $
      STRING(nochan(i+1),FORMAT='(I2)')
    hei(8+i/2)=WIDGET_LABEL(head_base, /ALIGN_LEFT, $
    VALUE=txt)
  ENDIF
ENDFOR
hei(16)=WIDGET_LABEL(head_base, /ALIGN_LEFT, $
VALUE='Delay [ms]: ' + $
STRING(1.0*adam_file_header.delay/ $
get_freq()*1000,FORMAT='(F10.2)'))
hei(17)=WIDGET_LABEL(head_base, /ALIGN_LEFT, $
VALUE='Acquisition frequency [Hz]: ' + $
STRING(get_freq(),FORMAT='(F10.2)'))
hei(19)=WIDGET_LABEL(head_base, /ALIGN_LEFT, $
VALUE='Acquisition time [ms]: ' + $
STRING(1.0*adam_file_header.length/ $
get_freq()*1000,FORMAT='(F10.2)'))
IF adam_file_header.samp EQ 1 THEN txt='Type of sampling: normal' $
ELSE txt='Type of sampling: quadrature'
hei(20)=WIDGET_LABEL(head_base, /ALIGN_LEFT, VALUE=txt)
hei(21)=WIDGET_TEXT(head_base, /WRAP, /FRAME, $
VALUE='Processing on data: ' + $
STRING(adam_file_header.prodesc), XSIZE=40, $
YSIZE=2, /SCROLL)
tr_data=WIDGET_BUTTON(head_base,VALUE='Transmitter data', $
EVENT_PRO='STATUS_BAR_event')
okb=WIDGET_BUTTON(head_base,VALUE='OK', $
EVENT_PRO='STATUS_BAR_event',UVALUE=head_base)
WIDGET_CONTROL, head_base, /REALIZE
XMANAGER, 'header_info',head_base

```

END

13.9 remove_info

Purpose Destroy all the info widgets if they are open.

Category widget function, function useable in the IDL[®] environment

Usage

NATO UNCLASSIFIED

- 96 -

SACLANTCEN M-127

NATO UNCLASSIFIED

`remove_info`

Common blocks This procedure uses the `INFO_Comm` common block to store information about the created widget.

Remarks This procedure is used when a new file is loaded.

Description This function destroys the widget if it is active.

```

PRO remove_info
COMMON INFO_Comm, pii, chi, tri, hei

  IF XREGISTERED('header_info') THEN $
    WIDGET_CONTROL, hei(0), /DESTROY
  IF XREGISTERED('Transmitter_info') THEN $
    WIDGET_CONTROL, tri(0), /DESTROY
  IF XREGISTERED('channel_info') THEN $
    WIDGET_CONTROL, chi(0), /DESTROY
  IF XREGISTERED('ping_info') THEN $
    WIDGET_CONTROL, pii(0), /DESTROY

```

END

13.10 transmitter_info

Purpose This routine shows the information on the transmitter device.

Category widget function, function useable in the IDL[®] environment

Usage

```
transmitter_info
```

Description See Subsection 13.8.

```

PRO transmitter_info
COMMON INFO_Comm, pii, chi, tri, hei

  IF XREGISTERED('Transmitter_info') THEN RETURN
  id=main_id()
  tri=lonarr(15)
  adam_file_header=get_header()
  IF !VERSION.ARCH EQ 'x86' THEN $
    trans_base=WIDGET_BASE(TITLE='Transmitter Info', /COLUMN, $

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

GROUP=id,TLB_FRAME_ATTR=1)$
ELSE trans_base=WIDGET_BASE(TITLE='Transmitter Info', /COLUMN, $
GROUP=id,TLB_FRAME_ATTR=11)
tri(0)=trans_base
CASE STRUPCASE(STRING(adam_file_header.pdesc)) OF
'TOPAS' : BEGIN
tri(1)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, /FRAME, $
VALUE='Topas transmitter data')
tri(2)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Interval [ms]: '+ $
STRING(adam_file_header.interval, FORMAT='(I10)'))
CASE adam_file_header.pulsform OF
0: txt='Te'
1: txt='Bu'
2: txt='Ri'
3: txt='Ch'
ELSE: txt='?'
ENDCASE
tri(3)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Pulse form: '+txt)
IF adam_file_header.HRP_comp EQ 1 THEN $
txt='Heave, roll, pitch compensation on' $
ELSE txt='Heave, roll, pitch compensation off'
tri(4)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE=txt)
tri(5)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Secondary frequency [Hz]: '+ $
STRING(adam_file_header.sec_freq, FORMAT='(I10)'))
tri(6)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Beam direction [deg]: '+ $
STRING(adam_file_header.beam_dir, FORMAT='(I10)'))
tri(7)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Scan sector of beam [deg]: '+ $
STRING(adam_file_header.scan_sec, FORMAT='(I10)'))
tri(8)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Scanning step [deg]: '+ $
STRING(adam_file_header.scan_step, FORMAT='(I10)'))
tri(9)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Beam pattern number: '+ $
STRING(adam_file_header.beam_pat, FORMAT='(I10)'))
tri(10)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, /FRAME, $
VALUE='Chirp data')
tri(11)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Chirp start [Hz]: '+ $
STRING(adam_file_header.chirp_start, FORMAT='(I10)'))
tri(12)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
VALUE='Chirp stop [Hz]: '+ $
STRING(adam_file_header.chirp_stop, FORMAT='(I10)'))
tri(13)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        VALUE='Chirp length [ms]:          '+ $
        STRING(adam_file_header.chirp_length, FORMAT='(I10)'))
    tri(14)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
        VALUE='Chirp type :                '+ $
        STRING(adam_file_header.chirp_type, FORMAT='(I10)'))
END
'SWATH': BEGIN
    tri(1)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, /FRAME, $
        VALUE='          Swath transitter data          ')
    tri(2)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
        VALUE='Gain start correction:          '+ $
        STRING(adam_file_header.gain_st,FORMAT='(I4)'))
    tri(3)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
        VALUE='Gain slope correction:          '+ $
        STRING(adam_file_header.gain_sl,FORMAT='(I4)'))
    tri(4)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
        VALUE='Pulse length correction:          '+ $
        STRING(adam_file_header.pulse_corr,FORMAT='(I4)'))
END
ELSE: tri(1)=WIDGET_LABEL(trans_base, /ALIGN_LEFT, $
    /FRAME, VALUE='No information about transmitter: '+ $
    STRING(adam_file_header.pdesc))
ENDCASE

okb=WIDGET_BUTTON(trans_base,VALUE='OK', $
    EVENT_PRO='STATUS_BAR_event',UVALUE=trans_base)
WIDGET_CONTROL, trans_base, /REALIZE
XMANAGER, 'Transmitter_info',trans_base

END

```

13.11 channel_info

Purpose This routine shows the information on the selected channel.

Category widget function, function useable in the IDL® environment

Usage

channel_info

Description See Subsection 13.8.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

PRO channel_info
COMMON INFO_Comm, pii, chi, tri, hei

    IF XREGISTERED('channel_info') THEN RETURN
    chi=lonarr(8)
    id=main_id()
    chan=get_channel_head()
    IF !VERSION.ARCH EQ 'x86' THEN $
        chan_base=WIDGET_BASE(TITLE='Channel Info', /COLUMN, $
            GROUP=id, TLB_FRAME_ATTR=1) $
    ELSE chan_base=WIDGET_BASE(TITLE='Channel Info', /COLUMN, $
        GROUP=id, TLB_FRAME_ATTR=11)
    chi(0)=chan_base
    chi(1)=WIDGET_LABEL(chan_base, /ALIGN_LEFT, /FRAME, $
        VALUE='          Channel number:      '+ $
        STRING(chan.no+1,FORMAT='(I10)'))
    freq=chan.freq*1.0
    gain=chan.gain*1.0
    sens=chan.sens
    range=chan.range
    IF get_version() EQ 1 THEN BEGIN
        sens=chan.sens*0.1
        range=chan.range*1.0
    ENDIF
    chi(2)=WIDGET_LABEL(chan_base, /ALIGN_LEFT, $
        VALUE='Signal frequency [Hz]:      '+ $
        STRING(freq,FORMAT='(F10.2)'))
    chi(3)=WIDGET_LABEL(chan_base, /ALIGN_LEFT, $
        VALUE='Total gain [dB]:            '+ $
        STRING(gain,FORMAT='(F10.2)'))
    chi(4)=WIDGET_LABEL(chan_base, /ALIGN_LEFT, $
        VALUE='Hydrophone sensitivity [bB]: '+ $
        STRING(sens, FORMAT='(F10.2)'))
    chi(5)=WIDGET_LABEL(chan_base, /ALIGN_LEFT, $
        VALUE='Type of hydrophone:         '+ $
        STRING(chan.hydtype,FORMAT='(I10)'))
    chi(6)=WIDGET_LABEL(chan_base, /ALIGN_LEFT, $
        VALUE='Input voltage range [V]:     '+ $
        STRING(range,FORMAT='(F10.2)'))
    chi(7)=WIDGET_TEXT(chan_base, /WRAP, /SCROLL, $
        VALUE='Description: '+ STRING(chan.desc), $
        XSIZE=23, YSIZE=5)
    okb=WIDGET_BUTTON(chan_base,VALUE='OK', $
        EVENT_PRO='STATUS_BAR_event',UVALUE=chan_base)
    WIDGET_CONTROL, chan_base, /REALIZE
    XMANAGER, 'channel_info',chan_base

END

```

NATO UNCLASSIFIED

- 100 -

13.12 change_channel_info

Purpose This procedure changes the information shown in the channel info widget.

Category widget function, function useable in the IDL[®] environment

Usage

Change_channel_info

Description This procedure only updates the values that are changed when the new ping is selected. The other parts of the channel information widget remain unchanged (see Subsection 13.11).

PRO change_channel_info

COMMON INFO_Comm, pii, chi, tri, hei

```

IF NOT XREGISTERED('channel_info') THEN RETURN
chan=get_channel_head()
WIDGET_CONTROL, chi(1), $
    SET_VALUE='          Channel number:      '+ $
    STRING(chan.no+1,FORMAT='(I10)')
freq=chan.freq*1.0
gain=chan.gain*1.0
sens=chan.sens
range=chan.range
IF get_version() EQ 1 THEN BEGIN
    sens=chan.sens*0.1
    range=chan.range*1.0
ENDIF
WIDGET_CONTROL, chi(2), $
    SET_VALUE='Signal frequency [Hz]:      '+ $
    STRING(freq,FORMAT='(F10.2)')
WIDGET_CONTROL, chi(3), $
    SET_VALUE='Total gain:                  '+ $
    STRING(gain,FORMAT='(F10.2)')
WIDGET_CONTROL, chi(4), $
    SET_VALUE='Hydrophone sensitivity [bB]: '+ $
    STRING(sens*0.1, FORMAT='(F10.2)')
WIDGET_CONTROL, chi(5), $
    SET_VALUE='Type of hydrophone:          '+ $
    STRING(chan.hydtype,FORMAT='(I10)')
WIDGET_CONTROL, chi(6), $
    SET_VALUE='Input voltage range [V]:     '+ $

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

        STRING(range,FORMAT='(F10.2)')
        WIDGET_CONTROL, chi(7), SET_VALUE='Description: ' + $
        STRING(chan.desc)

```

END

13.13 ping_info

Purpose This routine shows the information contained in the selected ping header.

Category widget function, function useable in the IDL[®] environment

Usage

```
ping_info
```

Description See Subsection 13.8.

```

PRO ping_info
COMMON INFO_Comm, pii, chi, tri, hei

month=[' ', 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', $
      'Sep', 'Oct', 'Nov', 'Dec']
IF XREGISTERED('ping_info') THEN RETURN
adam_file_pinghead=get_ping_head()
pii=lonarr(25)
id=main_id()
IF !VERSION.ARCH EQ 'x86' THEN $
    ping_base=WIDGET_BASE(TITLE='Ping Info', /COLUMN, $
        GROUP=id, TLB_FRAME_ATTR=1) $
ELSE ping_base=WIDGET_BASE(TITLE='Ping Info', /COLUMN, $
    GROUP=id, TLB_FRAME_ATTR=11)
pii(0)=ping_base
pii(1)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Date and time of ping acquisition: ' + $
        month(adam_file_pinghead.month) + ', ' + $
        STRING(adam_file_pinghead.day, FORMAT='(I2)') + ' ' + $
        STRING(adam_file_pinghead.hour, FORMAT='(I2)') + ':' + $
        STRING(adam_file_pinghead.min, FORMAT='(I2)') + ':' + $
        STRING(adam_file_pinghead.sec, FORMAT='(I2)') + '.' + $
        STRING(adam_file_pinghead.msec, FORMAT='(I3)') + ' ' + $
        STRING(adam_file_pinghead.year+1900, FORMAT='(I4)') )
IF adam_file_pinghead.dgps EQ 1 THEN txt='Differential GSP' $
ELSE txt='No differential GSP'
pii(2)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, /FRAME, $
    VALUE='                                Environmental data          ')

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

pii(3)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE=txt)
pii(4)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Latitude: ' + $
    STRING(adam_file_pinghead.lat_north, FORMAT='(F12.6)'))
pii(5)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Longitude: ' + $
    STRING(adam_file_pinghead.lon_east, FORMAT='(F12.6)'))
pii(6)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Zone: ' + $
    STRING(adam_file_pinghead.zone_lon, FORMAT='(F12.6)'))
pii(7)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, /FRAME, $
    VALUE='                Ship/Fish data                ')
pii(8)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Heading: ' + $
    STRING(adam_file_pinghead.heading, FORMAT='(F10.3)'))
pii(9)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Speed: ' + $
    STRING(adam_file_pinghead.speed, FORMAT='(F10.3)'))
pii(10)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Water depth: ' + $
    STRING(adam_file_pinghead.water_depth, FORMAT='(F10.3)'))
pii(11)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, /FRAME, $
    VALUE='                HPR data                ')
pii(12)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Tx Heave: ' + $
    STRING(adam_file_pinghead.tx_heave, FORMAT='(F10.4)'))
pii(13)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Tx Roll: ' + $
    STRING(adam_file_pinghead.tx_roll, FORMAT='(F10.4)'))
pii(14)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Tx Pitch: ' + $
    STRING(adam_file_pinghead.tx_pitch, FORMAT='(F10.4)'))
pii(15)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Rx Heave: ' + $
    STRING(adam_file_pinghead.rx_heave, FORMAT='(F10.4)'))
pii(16)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Rx Roll: ' + $
    STRING(adam_file_pinghead.rx_roll, FORMAT='(F10.4)'))
pii(17)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
    VALUE='Rx Pitch: ' + $
    STRING(adam_file_pinghead.rx_pitch, FORMAT='(F10.4)'))
IF get_version() EQ 2 THEN BEGIN
    pii(18)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, /FRAME, $
        VALUE='                Fish/buoy data                ')
    pii(19)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
        VALUE='F1 depth: ' + $
        STRING(adam_file_pinghead.f1_depth, FORMAT='(F10.4)'))
    pii(20)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $

```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

```

        VALUE='F1 xpos : '+ $
        STRING(adam_file_pinghead.f1_xpos, FORMAT='(F10.4)'))
    pii(21)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
        VALUE='F1 ypos : '+ $
        STRING(adam_file_pinghead.f1_ypos, FORMAT='(F10.4)'))
    pii(22)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
        VALUE='F2 depth: '+ $
        STRING(adam_file_pinghead.f2_depth, FORMAT='(F10.4)'))
    pii(23)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
        VALUE='F2 xpos : '+ $
        STRING(adam_file_pinghead.f2_xpos, FORMAT='(F10.4)'))
    pii(24)=WIDGET_LABEL(ping_base, /ALIGN_LEFT, $
        VALUE='F2 ypos : '+ $
        STRING(adam_file_pinghead.f2_ypos, FORMAT='(F10.4)'))
ENDIF
okb=WIDGET_BUTTON(ping_base,VALUE='OK', $
    EVENT_PRO='STATUS_BAR_event',UVALUE=ping_base)
WIDGET_CONTROL, ping_base, /REALIZE
XMANAGER, 'ping_info',ping_base

```

END

13.14 change_ping_info

Purpose This function changes the information shown in the ping info widget.

Category widget function, function useable in the IDL[®] environment

Usage

change_ping_info

Description See Subsection 13.12.

PRO change_ping_info

COMMON INFO_Comm, pii, chi, tri, hei

```

    month=['','Jan','Feb','Mar','Apr','May','Jun','Jul','Aug', $
        'Sep','Oct','Nov','Dec']
    IF NOT XREGISTERED('ping_info') THEN RETURN
    adam_file_pinghead=get_ping_head()
    WIDGET_CONTROL, pii(1), SET_VALUE= $
        'Date and time of ping acquisition: ' + $
        month(adam_file_pinghead.month) + ', ' + $
        STRING(adam_file_pinghead.day, FORMAT='(I2)') + ' ' + $
        STRING(adam_file_pinghead.hour, FORMAT='(I2)') + ':' + $

```

NATO UNCLASSIFIED

- 104 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```

        STRING(adam_file_pinghead.min, FORMAT='(I2)') + ':' + $
        STRING(adam_file_pinghead.sec, FORMAT='(I2)') + '.' + $
        STRING(adam_file_pinghead.msec, FORMAT='(I3)') + ' ' + $
        STRING(adam_file_pinghead.year, FORMAT='(I4)')
IF adam_file_pinghead.dgps EQ 1 THEN txt='Differential GSP' $
ELSE txt='No differential GSP'
WIDGET_CONTROL, pii(3), SET_VALUE=txt
WIDGET_CONTROL, pii(4), SET_VALUE='Latitude: ' + $
        STRING(adam_file_pinghead.lat_north, FORMAT='(F12.6)')
WIDGET_CONTROL, pii(5), SET_VALUE='Longitude: ' + $
        STRING(adam_file_pinghead.lon_east, FORMAT='(F12.6)')
WIDGET_CONTROL, pii(6), SET_VALUE='Zone: ' + $
        STRING(adam_file_pinghead.zone_lon, FORMAT='(F12.6)')
WIDGET_CONTROL, pii(8), $
        SET_VALUE='Heading: ' + $
        STRING(adam_file_pinghead.heading, FORMAT='(F10.3)')
WIDGET_CONTROL, pii(9), $
        SET_VALUE='Speed: ' + $
        STRING(adam_file_pinghead.speed, FORMAT='(F10.3)')
WIDGET_CONTROL, pii(10), $
        SET_VALUE='Water depth: ' + $
        STRING(adam_file_pinghead.water_depth, FORMAT='(F10.3)')
WIDGET_CONTROL, pii(12), SET_VALUE='Tx Heave: ' + $
        STRING(adam_file_pinghead.tx_heave, FORMAT='(F10.4)')
WIDGET_CONTROL, pii(13), SET_VALUE='Tx Roll: ' + $
        STRING(adam_file_pinghead.tx_roll, FORMAT='(F10.4)')
WIDGET_CONTROL, pii(14), SET_VALUE='Tx Pitch: ' + $
        STRING(adam_file_pinghead.tx_pitch, FORMAT='(F10.4)')
WIDGET_CONTROL, pii(15), SET_VALUE='Rx Heave: ' + $
        STRING(adam_file_pinghead.rx_heave, FORMAT='(F10.4)')
WIDGET_CONTROL, pii(16), SET_VALUE='Rx Roll: ' + $
        STRING(adam_file_pinghead.rx_roll, FORMAT='(F10.4)')
WIDGET_CONTROL, pii(17), SET_VALUE='Rx Pitch: ' + $
        STRING(adam_file_pinghead.rx_pitch, FORMAT='(F10.4)')
IF get_version() EQ 2 THEN BEGIN
    WIDGET_CONTROL, pii(19), SET_VALUE='F1 depth: ' + $
        STRING(adam_file_pinghead.f1_depth, FORMAT='(F10.4)')
    WIDGET_CONTROL, pii(20), SET_VALUE='F1 xpos : ' + $
        STRING(adam_file_pinghead.f1_xpos, FORMAT='(F10.4)')
    WIDGET_CONTROL, pii(21), SET_VALUE='F1 ypos : ' + $
        STRING(adam_file_pinghead.f1_ypos, FORMAT='(F10.4)')
    WIDGET_CONTROL, pii(22), SET_VALUE='F2 depth: ' + $
        STRING(adam_file_pinghead.f2_depth, FORMAT='(F10.4)')
    WIDGET_CONTROL, pii(23), SET_VALUE='F2 xpos : ' + $
        STRING(adam_file_pinghead.f2_xpos, FORMAT='(F10.4)')
    WIDGET_CONTROL, pii(24), SET_VALUE='F2 ypos : ' + $
        STRING(adam_file_pinghead.f2_ypos, FORMAT='(F10.4)')
ENDIF

```

NATO UNCLASSIFIED

NATO UNCLASSIFIED

SACLANTCEN M-127

END



NATO UNCLASSIFIED

14

Analysis of the signal: `op_rabel`

This section contains the routines used to process the signal. They are applied using the **Operation** submenu (see Section 2, Fig. 6) or the **Calculator** widget (see Section 15, Fig. 10).

All the functions and the procedures defined in this section act on the ping (if a single ping function is required), or on the selected ping range. Moreover, they work on the selected part of the signal, that means the part shown in the upper window.

All the operation functions obtain their signal from a handle past using the `h` parameter. The information contained in the handle are described in the former functions; these functions are used to create, copy, and delete the handle containing the data.

At the beginning of the functions implementing the “operations” there is a series of lines used to filter the data before performing the operations. The filter is called if the filter widget is displayed on the screen (Fig. 8).

14.1 `create()`

Purpose This function is used to copy the data into the handle structure used by the operation functions.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
h=create(v,name,xt,yt,op,FREQ=freq,MATRIX=matrix,STRUCT=struct)
```

Output The identifier of the handle containing the data.

Parameters

FREQ	is a parameter used to set the signal sampling frequency (spectrum and a_spectrum use this parameter)
MATRIX	is used as a flag (if set) to signal that the data are a matrix of points (used by some functions: image , swell)
STRUCT	is used to pass all the information of another handle using an info_s structure.

NATO UNCLASSIFIED

SACLANTCEN M-127

Arguments

v is the data vector
name is the string containing the name of the function that is performed (used by the operation functions)
xt and **yt** are the labels of abscissa and ordinate axis
op is the operation number: it is used by redraw to set the correct options for drawing the results.

Common blocks The file `op_rabel.com` contains the common block `HANDLE_Comm`. It contains two information structures: `info` and `junk`. `info` is the identifier of the handle which contain the information for the next plot; `junk` is an `info_s` structure, containing information about the last operation; this structure with the `HANDLE_Comm` common block structure is shown below:

```
COMMON HANDLE_Comm, info, junk
junk={info_s, OP:0, data:[0, 0, 0, 0], interval:[0,0], $
      id:0L, name:'', XT:'', YT:'', freq:0.0, type:0, baid:0L, $
      btid:0L, ptid:0L}
```

This is a description of the `info_s` structure:

OP is a flag used by redraw
name is the operation name
XT and **YT** are the axis names
freq is the signal sampling frequency
type is a flag that is set if **MATRIX** parameter is defined
data is used to store the actual parameter of the selected data (**ping**, **channel**, **card**, and **range**)
interval contains the indexes of the selected data vector
id is the identifier of the handle containing, as data, a matrix: the first column is composed by the **x** data, and the other columns the **y** data
baid is used by the **calculator functions**: it points to the base widget of the calculator where the operation information is stored
btid is used by the **calculator functions**: it points to the main button widget of the operation where the operation information is shown
ptid is used by the **calculator functions**: it points to the left hand button which contains an arrow or nothing depending on the last calculator operation.

Description The data are contained in two handles: the main one contains a structure of type `info_s`. In this structure is stored the identifiers of the handle containing the array of data.

NATO UNCLASSIFIED

The function frees the stored info handle. Then it fills the junk structure using the given parameters. If the **STRUT** parameter is set, this datum is used, or **junk** is filled using the data obtained by the **interval** and **get_data** functions. A handle is created where the data (past using the **v** argument) are stored; the handle identifier is then stored in the **junk** structure and a new handle is created where the **junk** structure is stored. The identifier of the new handle is stored in the common block variable **info** and returned by the function.

```
FORWARD_FUNCTION env, mean, get_swell, create, get_filter
```

```
FUNCTION create, v, name, xt, yt, op, FREQ=freq, $
  MATRIX=matrix, STRUCT=struct
@op_rabel.com
```

```
  free, info
  IF KEYWORD_SET(FREQ) THEN junk.freq=freq
  IF KEYWORD_SET(MATRIX) THEN junk.type=1
  IF KEYWORD_SET(STRUCT) THEN junk=struct $
  ELSE BEGIN
    tmp=interval()
    junk.interval=[MIN(tmp),MAX(tmp)]
    junk.data=get_data()
  ENDELSE
  junk.name=name
  junk.XT=xt
  junk.YT=yt
  junk.OP=op
  junk.id=HANDLE_CREATE(VALUE=v)
  info=HANDLE_CREATE(VALUE=junk)
  RETURN, info
```

```
END
```

14.2 free

Purpose This procedure is used to free the handles created using the **create** function.

Category operation routine, function not useable in the IDL® environment

Usage

```
  free, h
```

Arguments

NATO UNCLASSIFIED

SACLANTCEN M-127

h is the identifier of the handle containing the data information.

Description This procedure recovers the *h* handle information; then, it frees the handle containing the data matrix and the main handle.

```

PRO free, id

  IF KEYWORD_SET(ID) THEN $
    IF HANDLE_INFO(id) THEN BEGIN
      HANDLE_VALUE, id, inf
      HANDLE_FREE, inf.id
      HANDLE_FREE, id
    ENDIF
  ENDIF

END

```

14.3 get_handle

Purpose This procedure is used to recover the information of the last saved operation.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
get_handle, inf, vet [, ID=h]
```

Parameters

ID is the identifier of the handle containing the data information

Arguments

inf is a variable where the information structure of the last operation is returned

vet is a matrix where the data are stored

Common blocks This procedure uses the common block contained in the file `op_rabel.com`²⁸.

Description This procedure recovers the handle information using the information contained in the *ID* parameter or into the common block variable *info*. The information is returned using the *inf* and *vet* variables.

²⁸ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

- 110 -

```

PRO get_handle, inf, vet, ID=id
@op_rabel.com

  IF KEYWORD_SET(ID) NE 0 THEN BEGIN
    HANDLE_VALUE, id, inf
    HANDLE_VALUE, inf.id, vet
  ENDIF ELSE BEGIN
    IF KEYWORD_SET(INFO) THEN BEGIN
      HANDLE_VALUE, info, inf
      HANDLE_VALUE, inf.id, vet
    ENDIF ELSE inf=junk
  ENDELSE

END

```

14.4 energy

Purpose This procedure is used to calculate the signal energy.

Category operation routine, function not useable in the IDL[®] environment

Usage

energy, h [, /CLEAN]

Parameters

CLEAN when this keyword is set, the handle h is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Description This procedure calculates the energy of the signal in the upper graph ($s(t - t_0)$, $t_0 \geq t \leq t_1$) using the algorithm:

$$\text{energy} = \int_{t_0}^{t_1} s^2(\tau - t_0) d\tau$$

The integral is calculated using the five point Newton-Compton integration equation²⁹. The integration interval is the same as the one returned by the interval function.

²⁹It is used the IDL[®] function INT.TABULATED converted to a short name subroutine int_tabu.

NATO UNCLASSIFIED

SACLANTCEN M-127



```

PRO energy, h, CLEAN=clean

  get_handle, ID=h, inf, vet
  IF XREGISTERED('FILTER_WID') THEN BEGIN
    filter, get_filter(), h, /NOREDRAW
    get_handle, inff, vetf
    vet=TRANPOSE(vetf)
    X=vet(0,*)
    Y=vet(1,*)
  ENDIF ELSE BEGIN
    Y=vet(1,*)
    X=vet(0,inf.interval(0):inf.interval(1))
    Y=Y(inf.interval(0):inf.interval(1))-mean(Y)
  ENDELSE
  IF KEYWORD_SET(CLEAN) THEN free, h
  change_info, STRING(INT_TABULATED( X, Y^2))

END

```



14.5 copy

Purpose This procedure is used to create and display a duplicate of the selected upper window (it also adds information about ping number *etc.*).

Category operation routine, function not useable in the IDL[®] environment

Usage

copy, h [, /CLEAN]

Parameters

CLEAN when this keyword is set, the handle **h** is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³⁰.

Description This procedure creates a copy of the upper window data (used for storage and display).

³⁰ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

- 112 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```

PRO copy, h, CLEAN=clean
@op_rabel.com

  get_handle, ID=h, inf, vet
  IF KEYWORD_SET(CLEAN) THEN free, h
  IF inf.name EQ '' THEN BEGIN
    inf.name='copy'
    inf.XT='time [ms]'
    inf.YT='Amplitude [mV]'
    inf.OP=1
  ENDIF
  IF (inf.type EQ 0) OR (inf.name EQ 'copy') THEN BEGIN
    X=vect(0,inf.interval(0):inf.interval(1))
    Y=vect(1,inf.interval(0):inf.interval(1))
    tmp=create(TRANPOSE([X,Y]), inf.name, inf.XT, inf.YT, inf.OP)
  ENDIF ELSE tmp=create(TRANPOSE(vet), inf.name, inf.XT, $
    inf.YT, inf.OP, /MATRIX)
  redraw, /DOWN, /NOUP

END

```

14.6 a_energy

Purpose This procedure is used to calculate and plot the signal accumulated energy.

Category operation routine, function not useable in the IDL[®] environment

Usage

a_energy, h [, /CLEAN]

Parameters

CLEAN when this keyword is set, the handle **h** is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file op_rabel.com³¹.

³¹ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

Description This procedure calculates the normalized accumulated energy of the signal $(s(t - t_0), t_0 \leq t \leq t_1)$ in the upper graph using the algorithm:

$$a_energy(t - t_0) = \frac{\int_{t_0}^t s^2(\tau - t_0) d\tau}{\int_{t_0}^{t_1} s^2(\tau - t_0) d\tau}$$

The integrals are calculated with the rectangular approximation.

After the calculations, the lower graph is updated using redraw. The OP=1 means that a new two-dimensional graph needs to be printed. Also the **info button** is updated using the **energy** function.

◀

```
PRO a_energy, h, CLEAN=clean
```

```
@op_rabel.com
```

```
get_handle, ID=h, inf, vet
IF XREGISTERED('FILTER_WID') THEN BEGIN
    filter, get_filter(), h, /NOREDRAW
    get_handle, inff, vetf
    vetf=TRANPOSE(vetf)
    X=vetf(0,*)
    Y=vetf(1,*)
ENDIF ELSE BEGIN
    X=vet(0,inf.interval(0):inf.interval(1))
    Y=vet(1,*)
    Y=TRANPOSE(Y(inf.interval(0):inf.interval(1))-mean(Y))
ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
inf.interval=[0,inf.interval(1)-inf.interval(0)]
energy, create([X,Y], '', '', '', 0, STRUCT=inf)
tmp=Y^2
FOR i = 0, inf.interval(1) DO BEGIN
    Y(i)=TOTAL(tmp(0:i))
ENDFOR
Y=Y/MAX(Y)
tmp=create(TRANPOSE([X,Y]), 'a_energy', 'time [ms]', $
    'Normalized amplitude', 3, STRUCT=inf)
redraw, /DOWN, /NOUP
```

```
END
```

▶

14.7 envelope

Purpose This procedure is used to calculate and plot the signal envelope.

Category operation routine, function not useable in the IDL® environment

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

Usage

envelope, h [, /CLEAN]

Parameters

CLEAN when this keyword is set, the handle **h** is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³².

Description This procedure calculates the envelope of the signal $(s(t - t_0), t_0 < t < t_1)$ in the upper graph using the algorithm:

$$\text{envelope}(t - t_0) = \sqrt{\mathcal{H}(s(t - t_0))^2 + s^2(t - t_0)}$$

where $\mathcal{H}(s)$ is the Hilbert transformation of the signal s .

```

PRO envelope, h, CLEAN=clean
@op_rabel.com

get_handle, ID=h, inf, vet
IF XREGISTERED('FILTER_WID') THEN BEGIN
    filter, get_filter(), h, /NOREDRA
    get_handle, inff, vetf
    vet=TRANPOSE(vetf)
    X=vet(0,*)
    Y=vet(1,*)
ENDIF ELSE BEGIN
    X=vet(0,inf.interval(0):inf.interval(1))
    Y=vet(1,*)
    Y=TRANPOSE(Y(inf.interval(0):inf.interval(1))-mean(Y))
ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
Y=env(Y)
tmp=create(TRANPOSE([X,Y]), 'envelope', 'time [ms]', $
    'Amplitude [mV]', 1)
redraw, /DOWN, /NOUP

END

```

³² An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

14.8 a_envelope

Purpose This procedure is used to calculate and plot the signal envelope averaged on the range of pings.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
a_envelope, h [, /CLEAN]
```

Parameters

CLEAN when this keyword is set, the handle *h* is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file *op_rabel.com*³³.

Description This procedure calculates the envelope of the signals in the selected range following the same algorithm used in Subsection 14.7. Then it calculates a mean over the range. It is better to apply the swell filter to the signal before applying this function.

```
PRO a_envelope, h, CLEAN=clean
@op_rabel.com
```

```
get_handle, ID=h, inf, vet
IF (inf.name NE '') AND (inf.name NE 'copy') THEN BEGIN
    IF KEYWORD_SET(CLEAN) THEN free, h
    RETURN
ENDIF
IF XREGISTERED('FILTER_WID') THEN BEGIN
    filter, get_filter(), h, /NOREDRAW
    get_handle, inff, vetf
    vetf=TRANPOSE(vetf)
    X=vetf(0,*)
    Y=vetf(1:inf.data(3),*)
ENDIF ELSE BEGIN
    X=vet(0,inf.interval(0):inf.interval(1))
    Y=vet(1:inf.data(3),*)
    Y=Y(*,inf.interval(0):inf.interval(1))-[mean(Y)]# $
    REPLICATE(1.,inf.interval(1)-inf.interval(0)+1)
```

³³ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

- 116 -

```

ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
Y=REFORM(TOTAL(env(Y),1)/inf.data(3))
tmp=create(TRANPOSE([X,TRANPOSE(Y)]), 'a_envelope', $
'time [ms]', 'Amplitude [mV]', 1)
redraw, /DOWN, /NOUP

```

END

14.9 env()

Purpose This function is used to calculate the envelope of a data vector (it is used by the `envelope` and `a_envelope` routines).

Category operation routine, function useable in the IDL[®] environment

Usage

```
e=env(v)
```

Output The envelope of the input vector.

Arguments

`v` is the vector to be processed.

Description This procedure calculates the envelope of the signal using the algorithm described in Subsection 14.7.

```

FUNCTION env, v

```

```

    RETURN, ABS(SQRT(HILBERT(v)^2+v^2))

```

```

END

```

14.10 mean()

Purpose This function is used to calculate the mean value of a data vector.

Category operation routine, function useable in the IDL[®] environment

Usage

```
m=mean(v)
```

NATO UNCLASSIFIED

SACLANTCEN M-127

Output The mean of the input vector.

Arguments

v is the vector to be processed

Description This procedure calculates the mean of the whole signal (not only the part is shown in the upper window) using the algorithm:

$$\text{mean}(s) = \int_{t_i}^{t_f} s(\tau) d\tau$$

The integral is calculated using a total summation over the vector divided by the number of elements of the vector.

```
FUNCTION mean, v
```

```
  v1=REFORM(v)
  tmp=SIZE(v1)
  IF tmp(0) EQ 1 THEN BEGIN
    result=TOTAL(v)/tmp(tmp(0))
    RETURN, result(0)
  ENDIF ELSE RETURN, TOTAL(v,tmp(0))/tmp(tmp(0))
```

```
END
```

14.11 spectrum

Purpose This procedure is used to calculate and plot the signal spectrum.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
spectrum, h [, /CLEAN]
```

Parameters

CLEAN when this keyword is set, the handle h is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³⁴.

³⁴An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

Description This procedure calculates the power spectrum of the current ping using the formula:

$$|\mathcal{F}(s(t - t_0))|^2, f \geq 0$$

Where $\mathcal{F}(s)$ is the Fourier transform of the signal s . The abscissa values are assigned taking into account the sampling frequency using `freq` parameter in the `inf` structure. The spectrum is calculated only on the interval shown in the upper `Rabel` window.

```
PRO spectrum, h, CLEAN=clean
@op_rabel.com

get_handle, ID=h, inf, vet
IF XREGISTERED('FILTER_WID') THEN BEGIN
    filter, get_filter(), h, /NOREDRAW
    get_handle, inff, vetf
    vetf=TRANPOSE(vetf)
    Y=vetf(1,*)
ENDIF ELSE BEGIN
    Y=vet(1,*)
    Y=Y(inf.interval(0):inf.interval(1))-mean(Y)
ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
Y=2*(ABS(FFT(Y,1))^2.0)
tmpx=(inf.interval(1)-inf.interval(0)+1)/2
X=(inf.freq/2/tmpx/1000000.)*FINDGEN(tmpx+1)
Y=10*ALOG10(Y(0:tmpx))
tmpts=size(X)
inf.interval=[0, tmpts(tmpts(0))-1]
tmp=create([[X],[Y]], 'spectrum', 'Frequency [kHz]', $
    'Amplitude [dB]', 3, STRUCT=inf)
redraw, /DOWN, /NOUP
```

END

14.12 a_spectrum

Purpose This procedure is used to calculate and plot the signal spectrum averaged on the ping range.

Category operation routine, function not useable in the IDL[®] environment

Usage

a_spectrum, h [, /CLEAN]

NATO UNCLASSIFIED

SACLANTCEN M-127

Parameters

CLEAN when this keyword is set, the handle *h* is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file *op_rabel.com*³⁵.

Description This procedure calculates the power spectrum of each signal vector (with no mean) and then plots an average through the ping range. The spectrum definition may be found in Subsection 14.11.



```

PRO a_spectrum, h, CLEAN=clean
@op_rabel.com

get_handle, ID=h, inf, vet
IF (inf.name NE '') AND (inf.name NE 'copy') THEN BEGIN
  IF KEYWORD_SET(CLEAN) THEN free, h
  RETURN
ENDIF
IF XREGISTERED('FILTER_WID') THEN BEGIN
  filter, get_filter(), h, /NOREDRAW
  get_handle, inff, vetf
  vetf=TRANPOSE(vetf)
  X=vetf(0,*)
  Y=vetf(1:inf.data(3),*)
ENDIF ELSE BEGIN
  Y=vet(1:inf.data(3),*)
  Y=Y(*,inf.interval(0):inf.interval(1))-[mean(Y)]# $
  REPLICATE(1.,inf.interval(1)-inf.interval(0)+1)
ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
tmp=inf.interval(1)-inf.interval(0)+1
Y=2*(ABS(FFT(Y,1))^2.0)
tmpx=tmp/2
tmps=size(X)
inf.interval=[0, tmps(tmps(0))-1]
X=(inf.freq/2/tmpx/1000000.)*FINDGEN(tmpx+1)
Y=TOTAL(10*ALOG10(Y(*,0:tmpx)),1)/inf.data(3)
tmp=create([[X],[Y]], 'spectrum', 'Frequency [kHz]', $
  'Amplitude [dB]', 3, STRUCT=inf)
redraw, /DOWN, /NOUP

```

³⁵ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

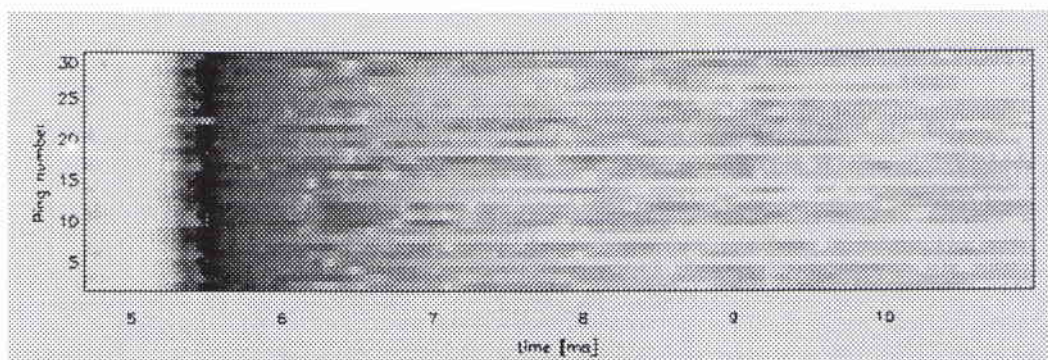


Figure 9: *The results of an image operation using the swell filter.*

END

14.13 image

Purpose This procedure is used to show an image of the signal envelope for all the signals in the ping range (see Fig. 9).

Category operation routine, function not useable in the IDL[®] environment

Usage

`image, h [, /CLEAN]`

Parameters

CLEAN when this keyword is set, the handle `h` is freed before exiting from the routine

Arguments

`h` is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³⁶.

Description This procedure calculates the envelope of each signal vector (with no mean) and shows the result on an IDL[®] TV scaled image (Fig. 9).

³⁶ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

```

PRO image, h, CLEAN=clean
@op_rabel.com

get_handle, ID=h, inf, vet
IF (inf.name NE '') AND (inf.name NE 'copy') THEN BEGIN
    IF KEYWORD_SET(CLEAN) THEN free, h
    RETURN
ENDIF
IF XREGISTERED('FILTER_WID') THEN BEGIN
    filter, get_filter(), h, /NOREDRAW
    get_handle, inff, vetf
    vet=TRANPOSE(vetf)
    X=vet(0,*)
    Y=vet(1:inf.data(3),*)
ENDIF ELSE BEGIN
    X=vet(0,inf.interval(0):inf.interval(1))
    Y=vet(1:inf.data(3),*)
    Y=Y(*,inf.interval(0):inf.interval(1))-[mean(Y)]# $
    REPLICATE(1.,inf.interval(1)-inf.interval(0)+1)
ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
Y=env(Y)
tmp=create([X,Y], 'image', 'time [ms]', 'Ping number',4,/MATRIX)
redraw, /DOWN, /NOUP

END

```

14.14 image_abs

Purpose This procedure is used to show an image of the signal absolute value for all the signals in the ping range.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
image_abs, h [, /CLEAN]
```

Parameters

CLEAN when this keyword is set, the handle h is freed before exiting from the routine

Arguments

h is the identifier of the handle used to pass the information to the function

NATO UNCLASSIFIED

- 122 -

SACLANTCEN M-127

NATO UNCLASSIFIED

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³⁷.

Description This procedure calculates the absolute value of each signal vector (with no mean) and then shows the result on a IDL[®] TV scaled image.

```

PRO image_abs, h, CLEAN=clean
@op_rabel.com

get_handle, ID=h, inf, vet
IF (inf.name NE '') AND (inf.name NE 'copy') THEN BEGIN
  IF KEYWORD_SET(CLEAN) THEN free, h
  RETURN
ENDIF
IF XREGISTERED('FILTER_WID') THEN BEGIN
  filter, get_filter(), h, /NOREDRAW
  get_handle, inff, vetf
  vet=TRANPOSE(vetf)
  X=vet(0,*)
  Y=vet(1:inf.data(3),*)
ENDIF ELSE BEGIN
  X=vet(0,inf.interval(0):inf.interval(1))
  Y=vet(1:inf.data(3),*)
  Y=Y(*,inf.interval(0):inf.interval(1))-[mean(Y)]# $
  REPLICATE(1.,inf.interval(1)-inf.interval(0)+1)
ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
Y=abs(Y)
tmp=create([X,Y], '|image|', 'time [ms]', 'Ping number',4,/MATRIX)
redraw, /DOWN, /NOUP

END

```

14.15 swell()

Purpose This procedure is used to set the swelling indexes of an image and show it.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
i=swell, h [, TIME=t] [,/CLEAN]
```

³⁷ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

Output If the T parameter is used, the function returns a vector containing the delay of each signal; otherwise, the vector of the sampling intervals, equivalent to the delays, is returned.

Parameters

CLEAN when this keyword is set, the handle h is freed before exiting from the routine
TIME is the sampling period

Arguments

h is the identifier of the handle used to pass the information to the function

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³⁸.

Description This function calculates the delay of each signals calculating the respective maximum and then finding the point where the signal is greater than one tenth of this maximum.

```

FUNCTION swell, h, TIME=time, CLEAN=clean
@op_rabel.com

  get_handle, ID=h, inf, vet
  IF (inf.name NE '') AND (inf.name NE 'copy') THEN BEGIN
    IF KEYWORD_SET(CLEAN) THEN free, h
    RETURN,0
  ENDIF
  IF KEYWORD_SET(TIME) THEN BEGIN
    IF XREGISTERED('FILTER_WID') THEN BEGIN
      filter, get_filter(), h, /NOREDRAW
      get_handle, inff, vetf
      vet=TRANPOSE(vetf)
      X=vet(0,*)
      Y=vet(1:inf.data(3),*)
    ENDIF ELSE BEGIN
      X=TRANPOSE(vet(0,inf.interval(0):inf.interval(1)))
      Y=vet(1:inf.data(3),*)
      Y=Y(*,inf.interval(0):inf.interval(1))-[mean(Y)]# $
      REPLICATE(1.,inf.interval(1)-inf.interval(0)+1)
    ENDELSE
  ENDIF ELSE BEGIN
    X=TRANPOSE(vet(0,inf.interval(0):inf.interval(1)))
    Y=vet(1:inf.data(3),*)
    Y=Y(*,inf.interval(0):inf.interval(1))-[mean(Y)]# $
    REPLICATE(1.,inf.interval(1)-inf.interval(0)+1)
  
```

³⁸ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

- 124 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```

ENDELSE
IF KEYWORD_SET(CLEAN) THEN free, h
d=inf.data
index=INTARR(d(3),/NOZERO)
FOR i=0, d(3)-1 DO BEGIN
    m = MAX(ABS(Y(i,*)))
    tmp = WHERE(ABS(Y(i,*)) GT m/10.0)
    index(i) = tmp(0)
ENDFOR
index=FLOOR(TOTAL(index)/d(3)) -index
IF KEYWORD_SET(TIME) THEN BEGIN
    X=INDGEN(d(3))+d(0)
    Y=index*time
    tmp=create([[X],[Y]], 'swell', 'Ping number', $
        'Delay time [ms]',5)
    redraw, /DOWN, /NOUP
ENDIF
RETURN, index
END

```

END

14.16 filter

Purpose This procedure is used to calculate and plot a filtered version of the signal.

Category operation routine, function not useable in the IDL[®] environment

Usage

```
filter, f, h, [, /CLEAN] [, /NOREDRAW]
```

Parameters

CLEAN when this keyword is set, the handle **h** is freed before exiting from the routine

NOREDRAW is a flag used to avoid drawing the results

Arguments

h is the identifier of the handle used to pass the information to the function

f is a **FILT_S** structure containing information about the filter type (Subsection 11.3).

Common blocks This procedure uses the common block contained in the file `op_rabel.com`³⁹.

³⁹ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

Description This procedure filters the selected signal. It could use two different algorithms of filtering: band-pass and matched filter. The band-pass filter is implemented using the IDL[®] function `DIGITAL_FILTER`; the matched filter is still not implemented. The swell and the zero mean filter can be applied to the data.

```
PRO filter, f, h, CLEAN=clean, NOREDRAW=noredraw, NOINTERVAL=nointerval
COMMON TMP_FILTER_Comm, index
@op_rabel.com
```

```
get_handle, ID=h, inf, vet
IF KEYWORD_SET(NOINTERVAL) THEN BEGIN
    oldinterval=inf.interval
    s=SIZE(vet)
    inf.interval=[0, s(s(0))-1]
ENDIF
X=vet(0,inf.interval(0):inf.interval(1))
IF inf.type EQ 1 THEN BEGIN
    Y=vet(1:inf.data(3),inf.interval(0):inf.interval(1))
    tmpy=vet(1:inf.data(3),*)
ENDIF ELSE BEGIN
    Y=vet(1,inf.interval(0):inf.interval(1))
    tmpy=vet(1,*)
ENDELSE
IF f.type(1) EQ 1 THEN Y=Y-[mean(tmpy)]#REPLICATE( $
    1.,inf.interval(1)-inf.interval(0)+1)
IF f.type(0) EQ 1 THEN BEGIN
    coeff = DIGITAL_FILTER(f.lf,f.hf,f.rpl,f.ord)
    tmp=FLTARR(inf.data(3),inf.interval(1)-inf.interval(0)+1)
    IF inf.type EQ 1 THEN $
        FOR i=0,inf.data(3)-1 DO $
            Y(i,*)=CONVOL(TRANSPPOSE(Y(i,*)),coeff,/EDGE_WRAP) $
    ELSE BEGIN
        tmp=CONVOL(TRANSPPOSE(Y),coeff,/EDGE_WRAP)
        Y=TRANSPPOSE(tmp)
    ENDELSE
ENDIF
s=SIZE(Y)
s=s(1)
IF (f.type(2) EQ 1) AND (s NE 1) THEN BEGIN
    inf1=inf
    vet=0
    inf1.interval=[0, inf.interval(1)-inf.interval(0)]
    index=swell(create([X, Y], $
        ', ', ', ', 0,/MATRIX,struct=inf1))
    IF inf.type EQ 0 THEN Y=SHIFT(Y,index(0)) $
    ELSE FOR i=0,inf.data(3)-1 DO $
```

NATO UNCLASSIFIED

- 126 -

SACLANTCEN M-127

NATO UNCLASSIFIED

```
        Y(i,*)=SHIFT(Y(i,*),index(i))
ENDIF
IF KEYWORD_SET(NOINTERVAL) THEN inf.interval=oldinterval
IF KEYWORD_SET(CLEAN) THEN free, h
tmp=create(TRANPOSE([X,Y]), 'filter', 'time [ms]', $
    'Amplitude [mV]', 1, STRUCT=inf)
IF (NOT KEYWORD_SET(NOREDRA)) THEN $
    redraw, /DOWN, /NOUP

END
```

NATO UNCLASSIFIED

SACLANTCEN M-127

15

Data calculation: `ca_rabel`

This section contains the routines used to displays a “calculator” widget. Using this widget it is possible to manipulate the data using buttons identified by different icons. Figure 10 show the appearance of the calculator when three data are in the stack.

The one-operand operation will act only on the data pointed by the pointer number **2**. When the operation should be performed on the actual data and range, it is possible to copy the data in the stack (using the copy button). The two-operands operator will act between arrow **1** and **2**. To move the pointer **1** it is necessary to click on one of the squared buttons. To move the pointer **2** click the long button on which the name of the operation and the [ping,channel,card,range] numbers are written.

The result of the operation is pushed on the stack using the push button identified by a right direction arrow. A left direction arrow can be used to pop out the last result.

Each operation is described by a different icon. A description of the icons follows (from left to right in Fig. 10):

- Push a left direction arrow
- Pop a right direction arrow
- Copy the two characters “Cp”
- Energy the two characters “ $\int E$ ”
- Accumulated energy the character “E”
- Envelope a rapidly variable function delimited by its envelope
- A_envelope the same logo then envelope with an “a” under it
- Power spectrum an “S” character
- A_Power spectrum the two characters “a”

NATO UNCLASSIFIED

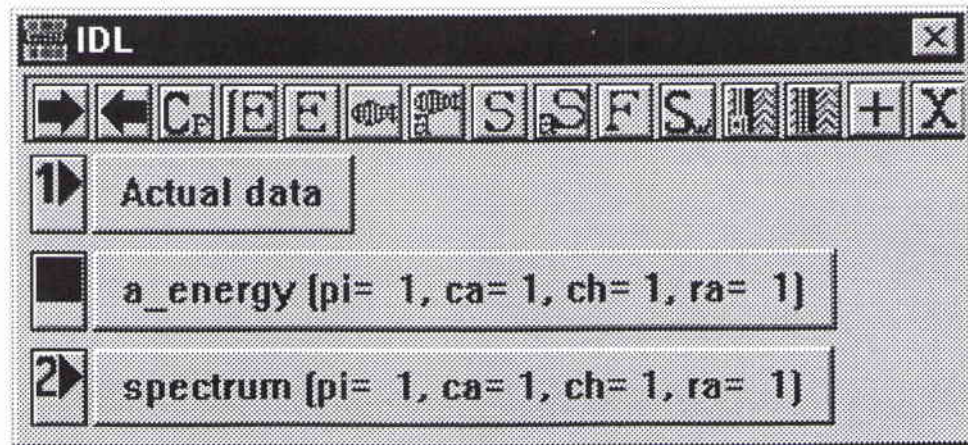


Figure 10: *The appearance of the calculator when only three data are in the stack.*

- Filter an “F” character
- Swell filter the two characters “Sw”
- |Image display| a schematic image logo with an absolute value simbol ($|\cdot|$) on the left
- Image display a schematic image logo
- Add the “+” sign
- Exit the “X” character

As an example, these are the steps to obtain Fig. 10:

1. Select, in the main window, ping 1, channel 1, card 1, and range 1
2. Start the calculator
3. Click on the Accumulated energy icon
4. Click on the Push icon
5. Click on the Copy icon
6. Click on the Spectrum icon

The fifth step is necessary because the spectrum operation will be otherwise performed on the last operation (the Accumulated energy one). Check it! Now, it is possible to see again the graphic of the Accumulated energy of the signal pushing the button where `a_energy (pi= 1 ...)` is written.

NATO UNCLASSIFIED

SACLANTCEN M-127

15.1 make_calculator

Purpose This procedure is used to display the calculator widget.

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
make_calculator [, GROUP=g]
```

Parameters

GROUP this parameter is used to pass the identifier of the main window to the widget creator: if the main widget is destroyed the calculator is also destroyed.

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. The indexes are used to set the **1** and **2** pointer (`INDEX1`, `INDEX2`) and to store the stack dimension (`INDEX3`). The calculator base widget identifier (`MAIN_CALC`) is also stored with the calculator stack (`ID_VET`) and its maximum dimension (`DIM`). Also the **1**, **2**, and null pointer icons are stored in this common block (`BMP271`, `BMP315`, `BMP528`).

```
COMMON CALC_Comm, MAIN_CALC, ID_VET, INDEX1, INDEX2, INDEX3, $
      DIM, BMP271, BMP315, BMP528
```

This function also uses the common block contained into the file `op_rabel.com`⁴⁰.

Description This function display the calculator. The button icons are contained in the file `ca_rabel.ico`. The calculator is created only if does not exist another calculator. The maximum dimension of the stack is given by the `DIM`. The `TLB_FRAME_ATTR` must be different for the Microsoft Windows with respect to X11. The first line of the widget is filled with buttons. The handles identifiers are stored in the `ID_VET` array. At the end of the function, the first two handles (filled with the information) are created. Then the `ca_rabel` function is called and the calculator widget realized (Fig. 10).

```
FORWARD_FUNCTION new_handle
```

```
PRO make_calculator, GROUP=Group
@op_rabel.com
@ca_rabel.com
@ca_rabel.ico
```

```
IF XREGISTERED('MAIN_CALC') THEN RETURN
```

⁴⁰ An explanation of its contents can be found in Subsection 14.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

DIM=10
INDEX1=0
INDEX3=1
INDEX2=1
ID_VET=LONARR(DIM)
IF N_ELEMENTS(Group) EQ 0 THEN GROUP=0
IF !VERSION.ARCH EQ 'x86' THEN $
    MAIN_CALC = WIDGET_BASE(GROUP_LEADER=Group, $
        COLUMN=1, MAP=1, TLB_FRAME_ATTR=9, $
        UVALUE='MAIN_CALC') $
ELSE MAIN_CALC = WIDGET_BASE(GROUP_LEADER=Group, $
    COLUMN=1, MAP=1, TLB_FRAME_ATTR=11, $
    UVALUE='MAIN_CALC')
BUTTON_BASE = WIDGET_BASE(MAIN_CALC, $
    ROW=1, FRAME=1, MAP=1, TITLE='BUTTON_BASE', $
    UVALUE='BUTTON_BASE')
BMPBTN21 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1689, $
    UVALUE='push')
BMPBTN22 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1742, $
    UVALUE='pop')
BMPBTN23 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1784, $
    UVALUE='copy')
BMPBTN1 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP973, $
    UVALUE='energy')
BMPBTN6 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1021, $
    UVALUE='a_energy')
BMPBTN9 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1115, $
    UVALUE='envelope')
BMPBTN11 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1204, $
    UVALUE='a_envelope')
BMPBTN17 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1481, $
    UVALUE='spectrum')
BMPBTN19 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1565, $
    UVALUE='a_spectrum')
BMPBTN16 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1432, $
    UVALUE='filter')
BMPBTN29 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1433, $
    UVALUE='swell')
BMPBTN20 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1355, $
    UVALUE='|image|')
BMPBTN22 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1616, $
    UVALUE='image')
BMPBTN25 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1884, $
    UVALUE='add')
BMPBTN28 = WIDGET_BUTTON( BUTTON_BASE,VALUE=BMP1969, $
    UVALUE='exit')
d=get_data()
junk1=junk
junk1.data=d

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

junk1.name='Actual data'
ID_VET(0)=HANDLE_CREATE(VALUE=junk1)
copy, create([TRANPOSE(xset()),$
    TRANPOSE(dataset())], '','',',',0)
tmp=create(TRANPOSE([TRANPOSE(xset()),$
    datamat()]], 'copy','Amplitude [mV]','Time [ms]',1,/MATRIX)
get_handle, junk, vet
junk.id=HANDLE_CREATE(VALUE=TRANPOSE(vet))
ID_VET(1)=HANDLE_CREATE(VALUE=junk)
ca_create
WIDGET_CONTROL, MAIN_CALC, /REALIZE
XMANAGER, 'MAIN_CALC', MAIN_CALC

```

END

15.2 set_calc

Purpose This procedure is used to set the calculator appearance after each operation.

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
set_calc
```

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure destroys the calculator widgets pointed by the handle stored in `ID_VET`. Then, it obtains information on the last performed operation and stores it in the `ID_VET` array and calls the `ca_create` function.

```

PRO set_calc
@ca_rabel.com

```

```

ca_destroy
get_handle, junk, vet
junk.id=HANDLE_CREATE(VALUE=TRANPOSE(vet))
free, ID_VET(INDEX3)
ID_VET(INDEX3)=HANDLE_CREATE(VALUE=junk)
ca_create

```

END

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

15.3 ca_create

Purpose This procedure is used to reconstruct the calculator using the common block information.

Category calculator routines, function not useable in the IDL[®] environment

Usage

ca_create

Common blocks The ca_rabel.com is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure re-creates the buttons inside the calculator widget. It creates the first line of buttons; then, it uses a FOR cycle to create all the other lines of data buttons. The data are recovered using the HANDLE_VALUE function. The first button, is created comparing the index (INDEX1, INDEX2, INDEX3) values with the line number is being created. The identifiers of the button are stored to be able, in the future, to destroy the buttons.

◀

PRO ca_create

@ca_rabel.com

```

HANDLE_VALUE, ID_VET(0), junk1
IF INDEX1 EQ 0 THEN val=BMP271 ELSE IF INDEX2 EQ 0 THEN val=BMP315 $
ELSE val=BMP528
junk1.baid = WIDGET_BASE(MAIN_CALC, ROW=1)
junk1.ptid = WIDGET_BUTTON( junk1.baid,VALUE=val, UVALUE='SS 0' )
val=junk1.name
junk1.btid = WIDGET_BUTTON( junk1.baid, VALUE=val, UVALUE='BB 0' )
HANDLE_VALUE, ID_VET(0), junk1, /SET
FOR i=1,INDEX3 DO BEGIN
    HANDLE_VALUE, ID_VET(i), junk1
    IF i EQ INDEX1 THEN val=BMP271 $
    ELSE IF i EQ INDEX2 THEN val=BMP315 $
    ELSE val=BMP528
    junk1.baid = WIDGET_BASE(MAIN_CALC, ROW=1)
    junk1.ptid = WIDGET_BUTTON( junk1.baid,VALUE=val, $
        UVALUE='SS'+STRING(i,FORMAT='(I2)') )
    val=junk1.name + $
        ' (pi='+ STRING(junk1.data(0),FORMAT='(I3)')+ $
        ', ca='+ STRING(junk1.data(1),FORMAT='(I2)') + $
        ', ch='+ STRING(junk1.data(2)+1,FORMAT='(I2)') + $
        ', ra='+ STRING(junk1.data(3),FORMAT='(I3)') + ' )'
    junk1.btid = WIDGET_BUTTON( junk1.baid, VALUE=val, $
        UVALUE='BB'+STRING(i,FORMAT='(I2)') )

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

        HANDLE_VALUE, ID_VET(i), junk1, /SET
    ENDFOR

```

END

15.4 ca_recreate

Purpose This procedure is used to redraw the calculator.

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
ca_recreate
```

Description The procedure destroys and recreates the calculator inside.

```

PRO ca_recreate

```

```

    ca_destroy
    ca_create

```

END

15.5 ca_destroy

Purpose This procedure is used to destroy the buttons in the calculator.

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
ca_destroy
```

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure destroys each group of buttons recalling the information stored in the handle.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

PRO ca_destroy
@ca_rabel.com

  FOR i=0,INDEX3 DO BEGIN
    HANDLE_VALUE, ID_VET(i), junk1
    WIDGET_CONTROL, junk1.baid, /DESTROY
  ENDFOR

END

```

15.6 ca_swap

Purpose This procedure is used to exchange the information contained in two consecutive buttons.

Category calculator routines, function not useable in the IDL[®] environment

Usage

`ca_swap`

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure exchanges the content of two consecutive indexes in the ID_VET. Then the `ca_recreate` function is called.

```

PRO ca_swap, id1, id2
@ca_rabel.com

  id=ID_VET(id2)
  ID_VET(id2)=ID_VET(id1)
  ID_VET(id1)=id
  ca_recreate

END

```

15.7 MAIN_CALC_Event

Purpose This procedure is used to manage the events generated by the calculator.

NATO UNCLASSIFIED

SACLANTCEN M-127

Category events manager routine, function not useable in the IDL[®] environment

Usage

```
MAIN_CALC_Event, ev
```

Arguments

ev is the event structure describing the event⁴¹.

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure obtains the event value and uses this information to call different functions. If the event is generated by a click on an **operation button**, the corresponding function is called (`a_energy`, for example). Moving the pointers generate a call to the `set_ptr` procedure. The use of the `new_handle()` function is important in calling the operation function. A new handle is created (adding some information if it is necessary) to furnish the data to the operation function; to delete this handle the operation function is called using the `CLEAN` keyword set.



```
PRO MAIN_CALC_Event, Event
@ca_rabel.com

WIDGET_CONTROL, Event.Id, GET_UVALUE=Ev
IF STRMID(Ev,0,2) EQ 'SS' THEN $
    set_ptr, 0, FIX(STRMID(Ev,2,4)) $
ELSE IF STRMID(Ev,0,2) EQ 'BB' THEN $
    set_ptr, 1, FIX(STRMID(Ev,2,4)) $
ELSE BEGIN
    WIDGET_CONTROL, /HOURLASS
    CASE Ev OF
        'energy': energy, new_handle(), /CLEAN
        'a_energy': BEGIN
            a_energy, new_handle(), /CLEAN
            set_calc
        END
        'envelope': BEGIN
            envelope, new_handle(), /CLEAN
            set_calc
        END
        'filter': BEGIN
            filter_wid
        END
        'spectrum': BEGIN
```

⁴¹ See the IDL[®] Reference Guide for each different event structures.

NATO UNCLASSIFIED

- 136 -

```

        spectrum, new_handle(FREQ=get_freq()), /CLEAN
        set_calc
    END
    'a_envelope': BEGIN
        a_envelope, new_handle(/MATRIX), /CLEAN
        set_calc
    END
    'a_spectrum': BEGIN
        a_spectrum, new_handle(/MATRIX,FREQ=get_freq()), /CLEAN
        set_calc
    END
    'image': BEGIN
        image, new_handle(/MATRIX), /CLEAN
        set_calc
    END
    '|image|': BEGIN
        image_abs, new_handle(/MATRIX), /CLEAN
        set_calc
    END
    'swell': BEGIN
        idx=swell(new_handle(/MATRIX), TIME=1/get_freq()), /CLEAN
        set_calc
    END
    'push': ca_push
    'pop': ca_delete
    'copy': ca_copy
    'add': ca_add
    'exit': ca_exit, Event
    ENDCASE
ENDELSE
END

```

15.8 ca_push

Purpose This procedure is used to push the last performed operation to the calculator stack.

Category calculator routines, function not useable in the IDL[®] environment

Usage

`ca_push`

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure copies the information contained in the last handle in the stack into a new handle. If the stack is full, the second element of the

NATO UNCLASSIFIED

SACLANTCEN M-127

stack is deleted and all the other lines of the stack are pushed up. Then the last two lines of the stack are swapped: the swap function also rebuilds the stack.

```

PRO ca_push
@ca_rabel.com

get_handle, junk, v
junk.id=HANDLE_CREATE(VALUE=TRANPOSE(v))
junk.baid = WIDGET_BASE(MAIN_CALC, ROW=1)
junk.ptid = WIDGET_BUTTON( junk.baid,VALUE=BMP528, $
    UVALUE='SS1')
junk.btid = WIDGET_BUTTON( junk.baid, $
    VALUE=junk.name + $
    ' (pi='+ STRING(junk.data(0),FORMAT='(I3)')+ $
    ', ca='+ STRING(junk.data(1),FORMAT='(I2)') + $
    ', ch='+ STRING(junk.data(2)+1,FORMAT='(I2)') + $
    ', ra='+ STRING(junk.data(3),FORMAT='(I3)') + ' )' )
INDEX2=INDEX2+1
INDEX3=INDEX3+1
IF INDEX3 EQ DIM THEN BEGIN
    INDEX3=INDEX3-1
    HANDLE_VALUE, ID_VET(1), junk1
    WIDGET_CONTROL, junk1.baid, /DESTROY
    free, ID_VET(1)
    FOR i=2, DIM-1 DO ID_VET(i-1)=ID_VET(i)
    IF INDEX2 GT INDEX3 THEN INDEX2=INDEX3
ENDIF
ID_VET(INDEX3)=HANDLE_CREATE(VALUE=junk)
ca_swap, INDEX3, INDEX3-1

END

```

15.9 ca_add

Purpose This procedure is used to draw two graphs indicated by the **1** and **2** pointers in the same drawing window.

Category calculator routines, function not useable in the IDL[®] environment

Usage

ca_add

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

Description This procedure draws the two pointed calculator data in the same drawing. The function has an effect only if the two graphs have the same number of points. This function is useful to compare different pings in the same context.

```

PRO ca_add
@ca_rabel.com
COMMON COLOR_Comm, COLORS, IND

  IF NOT KEYWORD_SET(IND) THEN BEGIN
    IND=1
    COLORS=!P.COLOR
  ENDIF
  IF INDEX1 EQ 0 THEN BEGIN
    tmp1=create([TRANSPOSE((xset()(interval())),$
      TRANSPOSE((dataset()(interval()))], ',' ',' ',' ',1)
    get_handle, ID=tmp1, inf1, vet1
  ENDIF ELSE BEGIN
    tmp1=new_handle(/PTR1)
    get_handle, ID=tmp1, inf1, vet1
  ENDELSE
  tmp2=new_handle()
  get_handle, ID=tmp2, inf2, vet2
  if inf1.interval(1) LT ((size(vet1))((size(vet1))(0))-1) THEN $
    vet1=vet1(*,inf1.interval(0):inf1.interval(1))
  if inf2.interval(1) LT ((size(vet2))((size(vet2))(0))-1) THEN $
    vet2=vet2(*,inf2.interval(0):inf2.interval(1))
  inf1.interval=[0,inf1.interval(1)-inf1.interval(0)]
  inf2.interval=[0,inf2.interval(1)-inf2.interval(0)]
  IF (inf1.interval(1) NE inf2.interval(1)) THEN RETURN
  tmp=create(TRANSPOSE(vet2), 'overposition', 'time [ms]', $
    'Amplitude [mV]', 10+inf1.OP, STRUCT=inf1)
  redraw, /DOWN, /NOUP
  tmp=create(TRANSPOSE(vet1), 'overposition', 'time [ms]', $
    'Amplitude [mV]', 5+inf1.OP, STRUCT=inf1)
  redraw, /DOWN, /NOUP
  tmp=create(TRANSPOSE(vet2), 'overposition', 'time [ms]', $
    'Amplitude [mV]', 5+inf1.OP, STRUCT=inf1)
  set_calc
END

```

15.10 ca_delete

Purpose This procedure is used to implement the pop function.

NATO UNCLASSIFIED

SACLANTCEN M-127

Category calculator routines, function not useable in the IDL[®] environment

Usage

`ca_delete`

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure clears the stack, frees the handle pointed by the 2 pointer, and moves up the following lines by one line⁴², then redraws the stack.

```

PRO ca_delete
@ca_rabel.com

  IF INDEX3 GT 1 THEN BEGIN
    ca_destroy
    free, ID_VET(INDEX2)
    FOR i=INDEX2+1,INDEX3 DO ID_VET(i-1)=ID_VET(i)
    INDEX3=INDEX3-1
    IF INDEX2 GT INDEX3 THEN INDEX2=INDEX3
    copy, ID_VET(INDEX2)
    ca_create
  ENDIF

END

```

15.11 `ca_copy`

Purpose This procedure is used to copy the upper window ping in the stack (included the actual data information: ping, channel, card, and range).

Category calculator routines, function not useable in the IDL[®] environment

Usage

`ca_copy`

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure plots the upper pointed ping and creates a data handle containing all the data matrix (if range is different from 1) and re-creates the stack. The function is useful to specify the data from which to obtain an image or on which to perform an operation.

⁴²If the stack has more than two lines.

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

PRO ca_copy
@ca_rabel.com

    copy, create([TRANSPPOSE(xset()),$
        TRANSPPOSE(dataset())], '', '', '', 0), /CLEAN
    tmp=create(TRANSPPOSE([TRANSPPOSE(xset()),$
        datamat()] ), 'copy', 'Amplitude [mV]', 'Time [ms]', 1, /MATRIX)
    set_calc

END

```

15.12 ca_exit

Purpose This procedure is used to destroy the calculator widget.

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
ca_exit
```

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure frees the handles in the stack and then destroys the calculator widget.

```

PRO ca_exit, Event
@ca_rabel.com

    HANDLE_FREE, ID_VET(0)
    FOR i=1, INDEX3 DO free, ID_VET(i)
    WIDGET_CONTROL, Event.TOP, /DESTROY

END

```

15.13 new_handle()

Purpose This function is used to create a new handler for operation and drawing purposes and to fill it with the contents of another one.

NATO UNCLASSIFIED

SACLANTCEN M-127

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
id=new_handle([FREQ=freq][,PTR1=ptr1][,MATRIX=matrix])
```

Parameters

FREQ is used to pass the value of the frequency to some functions
PTR1 is used to pass the handle to be copied (if different from the default stored in the common block)
MATRIX is a flag that must be set if a matrix of data is being stored in the handle

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure creates a new handle and fills it with the data contained in the **INDEX1** or in the **PTR1** position in the **ID_VET** vector. If the **FREQ** value is set it is possible to call the functions that need it (spectrum, etc.). The **MATRIX** flag is used if a matrix is being memorized.

```
FUNCTION new_handle, FREQ=freq, PTR1=ptr1, MATRIX=matrix
@ca_rabel.com

  IF KEYWORD_SET(PTR1) THEN $
    get_handle, ID=ID_VET(INDEX1), junk, vet $
  ELSE get_handle, ID=ID_VET(INDEX2), junk, vet
  IF KEYWORD_SET(FREQ) THEN junk.freq=freq
  junk.type=KEYWORD_SET(MATRIX)
  IF junk.interval(1) GT (SIZE(vet))(2) THEN $
    junk.interval=[0, junk.interval(1)-junk.interval(0)]
  junk.id=HANDLE_CREATE(VALUE=vet)
  RETURN, HANDLE_CREATE(VALUE=junk)
```

END

15.14 set_ptr

Purpose This procedure is used to manage the calculator 1 and 2 pointers.

Category calculator routines, function not useable in the IDL[®] environment

Usage

```
set_ptr, type, idx
```

NATO UNCLASSIFIED

- 142 -

Arguments

type is used to pass the type of pointer which must be moved;
idx is used to set the new pointer position.

Common blocks The `ca_rabel.com` is used to share the information about the calculator between the various functions. (Subsection 15.1.

Description This procedure sets the new value of **INDEX1** (or **INDEX2**). If the pointer **1** must be moved, it is necessary to click in the small square icon in the same column of the two pointers **1** and **2**. If the pointer **2** must be moved, you must click on the long button where the operation is specified. In the latter case when the pointer is moved, the lower drawing window will also be updated with the contents of the clicked button. This function deals also with the possibility that pointer **1** is placed over pointer **2** or vice versa. In such a case, if the stack has room the other pointer is pushed down or up. The calculator is then rebuilt.

```
PRO set_ptr, type, idx
@ca_rabel.com
```

```

CASE type OF
0: BEGIN
  INDEX1=idx
  IF INDEX2 EQ idx THEN BEGIN
    IF INDEX2 LT INDEX3 THEN INDEX2=INDEX2+1 $
    ELSE INDEX2=idx-1
    copy, ID_VET(INDEX2)
  END
END
1: BEGIN
  INDEX2=idx
  IF INDEX1 EQ idx THEN $
    IF INDEX1 GT 0 THEN INDEX1=INDEX1-1 $
    ELSE INDEX1=idx+1
  IF idx EQ 0 THEN $
    copy, create([TRANSPose(xset()),$
      TRANSPose(dataset())], 'Original data',$
      'Amplitude [mV]', 'time [ms]',1) $
    ELSE copy, ID_VET(idx)
  END
ENDCASE
ca_recreate
```

```
END
```

16

Adam file format specification, Version 2

This format was specified and released by Oddbjørn Bergem the 7th of May 1996.

16.1 Introduction

This document specifies the file format developed at SACLANTCEN for the **Abel** acquisition system: a system built to acquire data from triggered sonar systems at high data rates, and capable of storing 32 channels at 100kHz (16 channels at 200kHz) simultaneously.

The general idea is to store all necessary information in the file header, and to be flexible enough to use one file format for all the tasks required. At the same time, the format must be easy to implement and quick to read and store. The format consists of the following:

- An ID header which can be read independently from the version number and future changes.
- A general file header containing information which does not change during the acquisition. This header may be changed for later versions if required.
- A channel header describing each channel.
- A header containing information for each ping.
- The data acquired, stored sequentially.
- A short tail with a “checksum” and a description

Figure 11 shows the overall format.

16.2 Data types and sizes

The compiler should not use structure alignment: care has been taken to avoid any need of alignment. A C-compiler directive similar to

`#pragma nomember_alignment`

could be put inside the file before the declaration of the structures. To be portable between different platforms, it is important to keep the same structures sizes on each platform. The data structure sizes of the current version are:

- adamId: 12 bytes
- adamHead: 1424 bytes
- adamChanHead: 280 bytes
- adamPingHead: 368 bytes
- adamTail: 328 bytes

The data types have the following sizes:

- short = 2 bytes
- int = 4 bytes
- float = 4 bytes
- double = 8 bytes

16.3 The adamID

The purpose of this header is to describe the file format with version numbers and integer and floating point types. Due to possible different formats of the short, the high and low bytes of the adamId *short* variables must always be set equal. When these values are read, the results must be AND'ed with 0xFF.

The first field "tag" is a 4 bytes "magic number" corresponding to the ASCII values of the four letters "ADAM", i.e. [65 68 65 77] in decimal numbers.

The next field is the major version number (currently 2). The value of the field in decimal is therefore 0x0202.

The next two fields describe the integer and float format used in the file. For the "i.format", the following convention is used:

- i.format = 1 Big Endian, that is, multiple byte integers are stored beginning with the most significant byte first (*e.g.* HP, SUN)

NATO UNCLASSIFIED

SACLANTCEN M-127

- i_format = 2 Little Endian (*e.g.* Digital Alpha)

For the “f_format”, the following convention is used:

- f_format = 1 XDR(IEEE) format (*e.g.* HP, SUN)
- f_format = 2 Digital Alpha format

Other formats for the floating points may be added as needed.

The “dtype” field is a data type description which is free to be used.

```

typedef struct {
    char tag[4];           /* ‘‘Magic number’’: Ascii ‘ADAM’          */
    short version;         /* Current version = 2.                    */
    short f_format;        /* Float format                            */
    short i_format;        /* Integer format                          */
    short dtype;           /* Data type, free to use                  */
} adamId;

```

16.4 The adamHead

The adamHead has two purposes:

1. To give a place to store general information such as cruise information, and
2. To give enough information to read and interpret the data in the file.

The first block of information is called “Cruise info” and contains an ASCII description including the cruise name, the name of the experiment site *etc.* The next block, “Acquisition data”, contains important information both for the interpretation of the data and for the reading of the file. The fields and their meaning are:

- *sfreq* The sampling frequency in Hertz.
- *delay* The delay between the trigger and the start of acquisition. The delay is given in number of points. In order to obtain the delay in seconds one has to divide it by the sampling frequency.

NATO UNCLASSIFIED

- *length* The acquisition length given in number of points. The acquisition length in seconds for each trigger is then given by this number divided by the sampling frequency.
- *pings* The number of expected (or maximum) pings in the file. Because the acquisition can be interrupted, the number of actual pings in the file might be less than this number. The number of pings in the file can never be more than *pings*.
- *samp* This field describes the sampling technique used for the acquisition. *samp* = 1 means normal sampling, *samp* = 2 means in-phase and quadrature sampling (see Subsection 16.9).
- *noBits* The number of bits used for each sample in the data block. Currently only 16 is used.
- *noCards* The total number of cards used.
- *totChan* The total number of channels used on all cards.
- *noChan* This is an array which contains the number of channels used for each acquisition card. If a card is not used, the corresponding field should be set to 0: as a consequence the sum of each element of *noChan* should be equal to *totChan*.

The “Topas transmitter data” block and the “Swath transmitter data” block contain the transmitter parameters used for the respective systems. The “Processing” block is meant to be used to describe the post processing which is done on the data. Currently, this is free to use.

The “Other” block is left free for future requirements.

```
typedef struct {

/* Cruise info */
char cname[64];          /* Cruise name */
char sic[64];            /* Scientist in charge */
char fdesc[256];         /* Description of file, e.g. track number */
char site[64];           /* Experiment site */
char fname[64];          /* File name */
char pdesc[256];         /* Sonar system (e.g. TOPAS, Swath, etc.) */

/* Acquisition data */
float sfreq;             /* Sampling frequency in Hertz */
int delay;               /* Delay in number of points */
int length;              /* Acquisition length in points */
}
```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

int pings;           /* Maximum number of pings           */
int samp;            /* Sampling mode           */
int noBits;          /* Number of bits for each sample (now 16) */
int noCards;          /* Number of cards used (max 16) */
int totChan;          /* Total number of channels used */
int noChan[16];       /* Number of channels for each card */

/* Topas transmitter data */
short level;          /* Level (%) */
short interval;        /* Ping interval in mS */
short pulsform;        /* Type of pulse 0=Te, 1=Bu 2=Ri 3=Ch ?? */
short HRP_comp;        /* Heave, roll pitch compensation 1=used */
short sec_freq;        /* Secondary frequency */
short chirp_start;     /* Chirp/CW descriptions */
short chirp_stop;
short chirp_length;
short chirp_type;
short beam_dir;        /* Beam direction */
short scan_sec;        /* Scan sector of beam */
short scan_step;       /* Scanning step */
short beam_pat;        /* Beam pattern number */
short top_dummy[3];    /* Free to use */

/* Swath transmitter data */
int gain_st;           /* Gain start correction */
int gain_sl;           /* Gain slope correction */
int pulse_corr;        /* Pulse length correction */

/* Other transmitter data */
int trans[64];         /* Free variables */

/* Processing data */
int process;           /* Number describing post processing done */
char prodesc[256];     /* Verbal description of post processing */

} adamHead;

```

16.5 The *adamChanHead* header

This header contains a description for each channel. Accordingly, there are *adamHead.totChan* such headers. The fields contain the channel number, the total channel gain, the input voltage range *etc.* as described below.

```

typedef struct {

```

NATO UNCLASSIFIED

SACLANTCEN M-127

NATO UNCLASSIFIED

```

int    no;                /* Channel number                */
float freq;               /* Signal frequency in Hertz          */
float gain;               /* Total gain in dB applied to the channel*/
float sens;               /* Sensitivity of hydrophone in dB    */
int  hydtype;             /* Type of hydrophone used (free to use) */
float range;              /* Input voltage range in Volt (5V or 10V)*/
char desc[256];           /* Description of channel (filtering etc.)*/

} adamChanHead;

```

16.6 The adamPingHead header

This header is put at the beginning of each ping (each trigger). The intention is to supply information which might be changing between each ping, such as the time, speed, *etc.*

```

typedef struct {

/* Time stamps */
short year;                /* Start of acquisition info          */
short month;
short day;
short hour;
short min;
short sec;
short msec;

/* Environmental data */
short dgps;                /* 1 = differential GSP,
                           2 = No differential GPS          */
double lat_north;          /* E.g. 4404.0434                    */
double lon_east;           /* E.g. 0951.6325                    */
double zone_lon;           /* Free to use                      */
int zone_no;               /* Free to use                      */
float heading;             /* Ship heading in degrees           */
float speed;               /* Ship speed in knots               */
float water_depth;         /* Water depth in meters             */

/* Fish/buoy data (Intended for PUAPS) */
float f1_depth;            /* Depth below surface in meters     */
float f1_xpos;             /* Relative distance in meter        */
float f1_ypos;             /* Relative distance in meter        */
float f2_depth;            /* Depth below surface in meters     */
float f2_xpos;             /* Relative distance in meter        */
float f2_ypos;             /* Relative distance in meter        */

```

NATO UNCLASSIFIED

SACLANTCEN M-127

```

/* HPR info */
float tx_heave;          /* HPR at transmit time          */
float tx_roll;
float tx_pitch;
float rx_heave;          /* HPR at receive time          */
float rx_roll;
float rx_pitch;

/* Other info */
float angle;             /* Beam angle if applicable      */
float port;              /* Set to 1 if port beam (swath) */
float tv;                /* Swath start gain              */
float pre;               /* Swath pre gain                */
float tx_heading;        /* Heading of fish/buoy          */
float vari[60];          /* Free variables                */

} adamPingHead;

```

16.7 Data block format

The data block is a four dimensional array *data[Ping, Card, Sample, Channel]*. This means that the first word will be the data from the first ping, first card, first sample and the first channel. The next word will be the data from the first ping, first card, first sample and the second channel *etc.* The easiest way to understand how the data are stored is to look at Fig. 11. Using the notation in this figure, K,L,M,N can be computed as follows:

- K (the number of pings) must be computed based on the file size:

$$K = \frac{DFL - IHL - GHL - NOC \cdot CHL}{PHL + NOB \cdot LOS \cdot NOC}$$

where DFL is the data file length, IHL is the adamId size, GHL is the adamHead size, NOC is the number of used channel adamHead.totChan, CHL is the adamChanHead size, PHL is the adamPingHead size, NOB is the number of bytes used to store a data (adamHead.noBits/8), and LOS is the number of samples in an acquisition adamHead.length.

- L = adamHead.noCard
- M = adamHead.length
- N(i) = adamHead.noChan(i)

16.8 Tail

NATO UNCLASSIFIED

```

typedef struct {

short check;          /* Always set to 0xDADA          */
short suc;            /* Success. suc=0 OK,
                        suc<0 Errors,
                        suc>0 Warnings
int vari;             /* Free to use
char desc[256];       /* Verbal comments
                        should be added if suc!= 0
char desc_file[64];   /* Optional name of a comment file
} adamTail;

```

16.9 Sampling

Two types of sampling can be used:

1. Normal sampling
2. In-phase and quadrature sampling

For case 1), the samples are stored sequentially as a time series in the normal way. For case 2), the in-phase and quadrature components are stored as pairs. In this way, every n 'th sample will be the in-phase component, and every $(n+1)$ 'th component will be the quadrature component.

16.10 More information

When a field is left blank, it should be set to zero for integer and float, while the first position of an unused character field should be set to the null-termination character.

Index

A		F	
a_energy	113	filter	125
a_envelope	116	filter_wid	79
a_spectrum	119	FILTER_WID_Event	82
about_rabel	77	free	109
C		G	
ca_add	138	get_channel_head()	36
ca_copy	140	get_data()	23
ca_create	133	get_filter()	83
ca_delete	139	get_freq()	39
ca_destroy	134	get_handle	110
ca_exit	141	get_header()	35
ca_push	137	get_nochan()	41
ca_recreate	134	get_ping_head()	36
ca_swap	135	get_version()	38
change_channel_info	101	getr_ping_head()	37
change_info	93	H	
change_menu_bar	47	header_info	94
change_ping_info	104	help_rabel	76
change_status_bar	91	I	
channel_info	99	image	121
cm_rabel	54	image_abs	122
conv_head()	86	interval()	64
convert()	85	is_file_loaded()	33
copy	112	L	
create()	107	load_io_file	18
D		load_io_ni	19
data_file()	34	M	
datamat()	27	MAIN_CALC_Event	135
dataping()	29	main_id()	56
dataset()	25	MAIN_RABEL_Event	58
destroy_status_bar	92	main_w_create	50
E		main_w_destroy	53
energy	111	main_w_resize	53
env()	117	make_calculator	130
envelope	114	make_m_menu	44
exit_to_idl	77		

NATO UNCLASSIFIED

SACLANTCEN M-127

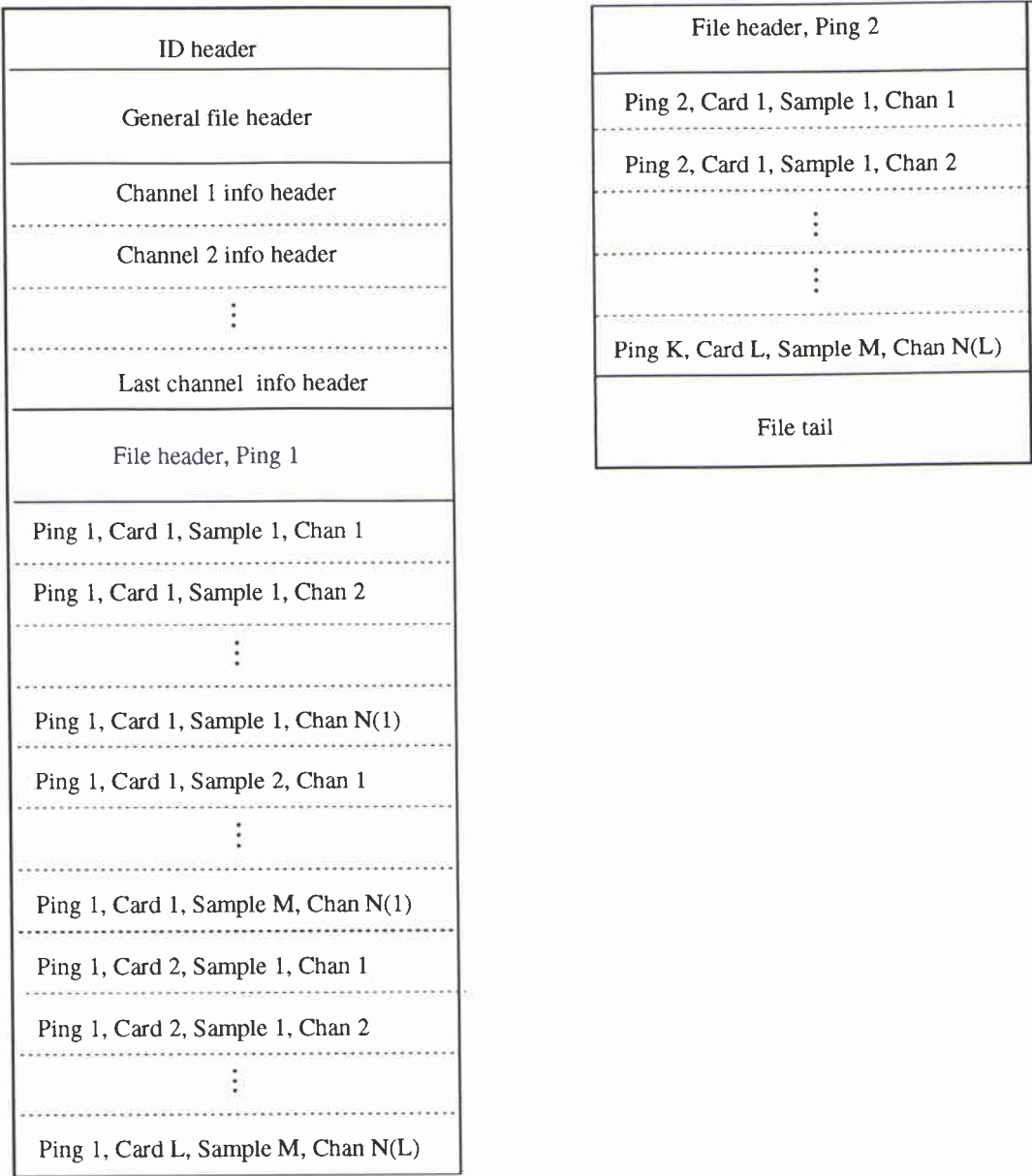


Figure 11: The overall format of an Adam data file.

NATO UNCLASSIFIED

SACLANTCEN M-127

make_status_bar 88
 mean() 117
 MENU_Event 59

N

n_of_byte() 40
 n_of_chan() 40
 n_of_ping() 31
 new_handle() 141

P

pan 65
 ping_info 102
 ping_slider_event 63
 print_graph 71
 PRINT_GRAPH_Event 74

Q

quit_rabel 56

R

rabel 16
 redraw 67
 remake_m_menu 45
 remove_info 96
 reset_zoom 64

S

save_menu_bar 47
 set_calc 132
 set_card_chan 33
 set_data 24
 set_ping 31
 set_ptr 142
 set_range 89
 SET_RANGE_Event 90
 set_slider 67
 set_title 66
 setfile 35
 spectrum 118
 STATUS_BAR_Event 92
 swell() 123

T

transmitter_info 97

X

xset() 26

Z

zoom_event 61

NATO UNCLASSIFIED

Document Data Sheet

NATO UNCLASSIFIED

Security Classification NATO UNCLASSIFIED		Project No. 033-2
Document Serial No. M-127	Date of Issue May 1997	Total Pages 162 pp.
Author(s) Canepa, G.		
Title Rabel – a program to replay Abel. Version 2.2		
Abstract The description of a graphics software tool used to view and manipulate data files is presented. The software is implemented to run in the IDL® environment on any platform. Rabel is able to read data from files in the ADAM format, a SACLANTCEN standard used to save data acquired using the ABEL system. The software has a sophisticated graphic user interface which simplifies loading and scanning of data files. The program is also able to perform simple operations on the data: enveloping, zooming, filtering, etc. This report is intended as a detailed description of the implementation at a source code level. A short introduction to the program capability is given.		
Keywords IDL routine – graphic packages – data file format – data analysis		
Issuing Organization North Atlantic Treaty Organization SACLANT Undersea Research Centre Viale San Bartolomeo 400, 19138 La Spezia, Italy [From N. America: SACLANTCEN CMR-426 (New York) APO AE 09613]		Tel: +39 (0)187 540 111 Fax: +39 (0)187 524 600 E-mail: library@saclantc.nato.int

NATO UNCLASSIFIED

Initial Distribution for M-127

SCNR for SACLANTCEN

SCNR Belgium	1
SCNR Canada	1
SCNR Denmark	1
SCNR Germany	1
SCNR Greece	2
SCNR Italy	1
SCNR Netherlands	1
SCNR Norway	1
SCNR Portugal	1
SCNR Spain	1
SCNR Turkey	1
SCNR UK	1
SCNR US	2
French Delegate	1
SECGEN Rep. SCNR	1
NAMILCOM Rep. SCNR	1

SACLANT 3

National Liaison Officers

NLO Canada	1
NLO Denmark	1
NLO Germany	1
NLO Italy	2
NLO Netherlands	1
NLO UK	3
NLO US	4

Total external distribution	34
SACLANTCEN Library	26

Total number of copies 60