

*Domain/OS Call  
Reference  
(Volume 2)*

012888-A00

apollo

# Domain/OS Call Reference

## Volume 2

Order No. 012888-A00

Apollo Computer Inc.  
330 Billerica Road  
Chelmsford, MA 01824

Confidential and Proprietary. Copyright © 1988 Apollo Computer, Inc., Chelmsford, Massachusetts.  
Unpublished — rights reserved under the Copyright Laws of the United States. All rights reserved.

First Printing: June 1988

Apollo and Domain are registered trademarks of Apollo Computer Inc.

UNIX is a registered trademark of AT&T in the USA and other countries. APPLE and LaserWriter are registered trademarks of Apple Computer, Inc. POSTSCRIPT is a registered trademark of Adobe Systems Inc. TEKTRONIX is a registered trademark of Tektronix, Inc. GENICOM is a trademark of Genicom Corporation. Spinwriter is a registered trademark of NEC Corporation.

3DGM, Aegis, D3M, DGR, Domain/Access, Domain/Ada, Domain/Bridge, Domain/C, Domain/ComController, Domain/CommonLISP, Domain/CORE, Domain/Debug, Domain/DFL, Domain/Dialogue, Domain/DQC, Domain/IX, Domain/Laser-26, Domain/LISP, Domain/PAK, Domain/PCC, Domain/PCI, Domain/SNA, Domain X.25, DPSS, DPSS/Mail, DSEE, FPX, GMR, GPR, GSR, NLS, Network Computing Kernel, Network Computing System, Network License Server, Open Dialogue, Open Network Toolkit, Open System Toolkit, Personal Supercomputer, Personal Super Workstation, Personal Workstation, Series 3000, Series 4000, Series 10000, and VCD-8 are trademarks of Apollo Computer Inc.

Apollo Computer Inc. reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should in all cases consult Apollo Computer Inc. to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF APOLLO COMPUTER INC. HARDWARE PRODUCTS AND THE LICENSING OF APOLLO COMPUTER INC. SOFTWARE PROGRAMS CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN APOLLO COMPUTER INC. AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY APOLLO COMPUTER INC. FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY BY APOLLO COMPUTER INC. WHATSOEVER.

IN NO EVENT SHALL APOLLO COMPUTER INC. BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATING TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF APOLLO COMPUTER INC. HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

THE SOFTWARE PROGRAMS DESCRIBED IN THIS DOCUMENT ARE CONFIDENTIAL INFORMATION AND PROPRIETARY PRODUCTS OF APOLLO COMPUTER INC. OR ITS LICENSORS.

This manual is dedicated to the memory of Kriss Kellerman,  
a valued friend and colleague.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document further explains that regular reconciliation of accounts is essential to identify any discrepancies early on and prevent them from escalating into larger issues.

In addition, the document highlights the need for transparency and accountability in financial reporting. It suggests that all stakeholders should have access to the relevant information and that the reporting process should be clear and concise. This helps in building trust and ensuring that everyone is on the same page regarding the company's financial health.

The second part of the document focuses on budgeting and financial planning. It provides a detailed guide on how to create a realistic budget that takes into account all potential risks and opportunities. The document also discusses various financial ratios and metrics that can be used to evaluate the company's performance and make informed decisions about future investments and operations.

Finally, the document concludes with a series of practical tips and best practices for managing the company's finances effectively. These include maintaining a strong relationship with financial institutions, staying up-to-date on tax regulations, and regularly reviewing the financial strategy to ensure it remains aligned with the company's long-term goals.

## Preface

The *Domain/OS Call Reference* describes unique programming interfaces to Domain@/OS available only on a Domain system. The interfaces documented here can be called from all three Domain/OS environments, and, except where noted, can be used freely together with UNIX\* system and library calls. For instance, the stream IDs returned by the Domain/OS call `ios_$open` are identical to the file descriptors expected by the UNIX system calls `read` or `write`, and can even be passed to `fdopen` to obtain the file pointers used by the standard I/O library.

The declarations of Domain/OS interfaces are grouped by function into insert files. Each call and each predefined data type, variable, or constant has a prefix that generally identifies the insert file in which it's declared. Interfaces prefixed with `pad_$`, for example, allow programs to manipulated pads and windows on the display via the Display Manager. They are declared in the following insert files:

<code>/usr/include/apollo/pad.h</code>	for C
<code>/sys/ins/pad.ins.pas</code>	for Pascal
<code>/sys/ins/pad.ins.ftn</code>	for FORTRAN

Some extremely common data types and constants are defined in the "base" insert file for each language. The base insert files are

---

\* UNIX is a registered trademark of AT&T in the USA and other countries.

<code>/usr/include/apollo/base.h</code>	for C
<code>/sys/ins/base.ins.pas</code>	for Pascal
<code>/sys/ins/base.ins.ftn</code>	for FORTRAN

For example, the Domain/OS completion status type, `status_t`, the system clock value type, `time_clock_t`, and the stream ID type, `ios_id_t`, as well as the values for the standard streams, `ios_stdin`, `ios_stdout`, and `ios_stderr`, are all defined in the base insert file.

The *Domain/OS Call Reference* is divided into sections reflecting the grouping of interface declarations into insert files. Information about an interface can usually be found in the section whose name is the prefix of that interface. The sections are ordered alphabetically by prefix.

Each section begins with an introduction to the interfaces it documents. The introduction contains a brief discussion of the function of the interfaces, and explains any predefined constants and data types. The introduction is followed by an individual description of each call ordered alphabetically by the name of the call.

Each section is preceded by a "Chapter Table of Contents" on colored stock to speed location of individual call descriptions.

---

## Call Descriptions

All call descriptions and introductions use a common format consisting of the following parts:

- The **NAME** part contains the name of the Domain/OS call and a brief description of its purpose. The introductory sections all use the name "intro".
- The **SYNOPSIS** parts list the include files recommended for the call, followed by a declaration for the call or an example of its use. The **SYNOPSIS** parts of an introduction just show the include files that define the interface. Nearly all call descriptions contain three **SYNOPSIS** sections — one each for C, Pascal, and FORTRAN. C and Pascal synopses contain declarations of the call and its arguments in the appropriate language.

Because FORTRAN lacks user-defined data types, a FORTRAN synopsis does not attempt to present a canonical declaration for the call it describes. Instead, a FORTRAN synopsis contains an example of declaring the necessary arguments and making the call in FORTRAN, and, by necessity, sometimes contains extra variables with arbitrary values chosen only for the example. Note that a FORTRAN **SYNOPSIS** is just one example of how to make the call in FORTRAN, and cannot be the "best" example for every application.

- The **DESCRIPTION** part describes the purpose of the call, the use of its arguments, and the value it returns, if any.
- The **EXAMPLES** part contains examples of using the call.
- The **FILE** part contains the pathnames of objects used by the call.
- The **NOTES** part contains additional helpful information that is not exclusively descriptive of the call itself. **NOTES** may explain how a closely related call differs, or point out problems with using the call in combination with others.
- The **SEE ALSO** part contains references to related calls and introductions.

---

## Online Access

All of the information contained in this document is available online via the Aegis **help** command and is integrated into the UNIX online manual accessible via **man**.

In the Aegis environment, you can obtain information on a call by typing:

```
$ help calls name_of_call
```

where *name\_of\_call* is the name of the call exactly as it should appear in a program. To get information on **ios\_\$open**, for instance, type

```
$ help calls ios_$open
```

You can read the appropriate introduction two ways, by either typing only the prefix, or by typing *prefix\_\$intro*. For instance, either of the two following command lines will get the introduction to the **ios\_\$** calls:

```
$ help calls ios
```

```
$ help calls ios_$intro
```

In a UNIX environment, you can obtain information on a call by typing:

```
$ man a name_of_call
```

where *name\_of\_call* is the name of the Domain/OS call with the dollar sign (\$) removed from the prefix. (The dollar sign was omitted so arguments to **man** do not have to be quoted to avoid variable substitution in UNIX shells.) To get information on **ios\_\$open**, for instance, type

```
$ man a ios_open
```

You can read the appropriate introduction two ways, by either typing only the prefix, or by typing *prefix\_intro*. For instance, either of the two following command lines will get the introduction to the **ios\_\$** calls:

\$ man a ios

\$ man a ios\_intro

Using an "a" for the section specifier to **man** restricts the manual search to Domain/OS call documentation and may be omitted if such a restriction is not necessary.

---

## Related Manuals

The file `/install/doc/apollo/os.v.10.n_manuals` lists current titles and revisions for all manuals available at update *n* of SR10. For example, at SR10.0 you can check `/install/doc/apollo/os.v.10.n_manuals` to make sure you have the latest version of the manuals you use, or to find additional manuals you might need. (In the Aegis environment, you can access the same information by typing "help manuals".)

Refer to the *Domain Documentation Quick Reference* (002685) and the *Domain Documentation Master Index* (011242) for a complete list of related documents. For more information on programming in Domain/OS, refer to the following documents:

- *SysV Programmer's Reference* (005799)
- *BSD Programmer's Reference* (005801)
- *Programming with Domain/OS Calls* (005506)
- *Programming with Domain/OS Calls for IPC* (005696)
- *Domain/OS Programming Environment Reference* (011010)
- *Domain Distributed Debugging Environment Reference* (011024)
- *Domain Binder and Librarian Reference* (004977)
- *Domain C Language Reference* (002093)
- *Domain Pascal Language Reference* (000792)
- *Domain FORTRAN Language Reference* (000530)

---

## Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. To make it easy for you to communicate with us, we provide the Apollo® Product Reporting (APR) system for comments related to hardware, software, and documentation. By using this formal channel you make it easy for us to respond to your comments.

You can get more information about how to submit an APR by consulting the appropriate Command Reference manual for your environment (Aegis™, BSD, or SysV). Refer to the `mkapr` command description. You can view the same description online by typing:

`$ help mkapr` (in the Aegis environment)

`$ man 1 mkapr` (in a UNIX environment)

Alternatively, you may use the Reader's Response form at the back of this manual to submit comments about the manual.

---

## Documentation Conventions

This manual uses the following symbolic conventions:

- |                         |  |
|-------------------------|--|
| <i>literal symbols</i>  | Keywords and predefined symbols are printed in bold type. They must be used exactly as documented.   |
| <i>variable symbols</i> | Placeholders for symbols that can be chosen by the programmer are printed in italics. For example, the names chosen for call arguments are printed in italics. |
| <i>examples</i>         | Examples of program code appears in a constant width font to preserve indentation.   |
|                         | A vertical bar separates items in a list of choices.   |

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, accounts payable, and accounts receivable. It also outlines the procedures for recording these transactions, including the use of double-entry bookkeeping to ensure that the books balance.

The second part of the document focuses on the analysis of the financial data. It explains how to calculate key financial ratios and metrics, such as the gross profit margin, operating profit margin, and return on investment. These calculations are essential for understanding the company's financial performance and identifying areas for improvement. The document also discusses the importance of comparing the company's performance to industry benchmarks and providing a clear explanation of the reasons for any variances.

The final part of the document covers the preparation of financial statements. It provides a step-by-step guide to creating the income statement, balance sheet, and cash flow statement. It also discusses the importance of auditing the financial statements to ensure their accuracy and reliability. The document concludes by emphasizing the role of financial reporting in decision-making and the overall success of the business.

# Contents

---

## aclm

intro .....	change the protected subsystem privilege level
aclm_\$down .....	deassert subsystem manager rights
aclm_\$up .....	assert subsystem manager rights

---

## cal

intro .....	the Domain/OS calendar service
cal_\$add_clock .....	add two clock values
cal_\$apply_local_offset .....	convert a UTC clock value to a local one
cal_\$clock_to_sec .....	convert clock ticks into whole seconds
cal_\$cmp_clock .....	compare two clock values
cal_\$decode_ascii_date .....	decode an ASCII date
cal_\$decode_ascii_time .....	decode an ascii time
cal_\$decode_ascii_tzdiff .....	decode an ASCII time zone
cal_\$decode_local_time .....	get the local date and time
cal_\$decode_time .....	decode a clock value
cal_\$encode_time .....	encode a date and time
cal_\$float_clock .....	convert clock ticks into seconds
cal_\$get_info .....	get the local time zone
cal_\$get_local_time .....	get the local time
cal_\$remove_local_offset .....	convert local time to UTC
cal_\$sec_to_clock .....	convert seconds to a clock value
cal_\$sub_clock .....	subtract two clock values
cal_\$weekday .....	return the weekday for a date
cal_\$write_timezone .....	set the local time zone

---

## ctm

intro .....	Color Table Manager
ctm_\$alloc_pv .....	allocate unused pixel values
ctm_\$find_color .....	find a color in the color map
ctm_\$inc_use_count .....	increment pixel value use counts
ctm_\$mark_read_only .....	share pixel values with other processes
ctm_\$release_pv .....	decrement pixel value use counts

---

## ec2

intro ..... Level 2 Eventcount Manager  
ec2\_Sadvance ..... increment an eventcount  
ec2\_Sinit ..... initialize an eventcount  
ec2\_Sread ..... return an eventcount value  
ec2\_Swait ..... wait for an eventcount trigger  
ec2\_Swait\_slow\_io ..... wait for an eventcount trigger during I/O  
ec2\_Swait\_svc ..... wait for an eventcount trigger or asynchronous fault

---

## error

intro ..... Domain/OS error reporting  
error\_Scode ..... return the error code from a completion status  
error\_Sfail ..... test the fail bit in a completion status  
error\_Sfind\_text ..... find error reporting text  
error\_Sget\_text ..... get error reporting text  
error\_Sinit\_std\_format ..... initialize standard error reporting  
error\_Smodule ..... return the module code from a completion status  
error\_Sprint ..... print an error message  
error\_Sprint\_format ..... print an error message in standard format  
error\_Sprint\_name ..... print error text with a name  
error\_Sstd\_format ..... print a standard error message  
error\_Ssubsys ..... return the subsystem code from a completion status

---

## fault

intro ..... Domain/OS fault codes

---

## fpp

intro ..... the Floating-Point Package  
fpp\_Scontrol ..... modify the floating-point control register  
fpp\_Srestore ..... load a new floating-point state  
fpp\_Ssave ..... save the floating-point state  
fpp\_Ssave\_restore ..... save the current and load a new floating-point state  
fpp\_Ssave\_restore\_size ..... return the size of a floating-point state  
fpp\_Sstatus ..... modify the floating-point status register

---

## gmf

intro	using graphics metafiles
gmf_Sclose	close a GMF
gmf_Scopy_plane	copy a bitmap plane to a GMF
gmf_Scopy_subplane	copy part of a bitmap plane to a GMF
gmf_Sopen	open a GMF
gmf_Srestore_plane	restore a bitmap from a GMF

---

## ios

intro	device-independent I/O
ios_Schange_path_name	change an object's pathname
ios_Sclose	close a stream
ios_Screate	create an object and open a stream to it
ios_Sdelete	delete an object
ios_Sdup	duplicate a stream
ios_Sequal	determine if two streams are open on the same object
ios_Sforce_write_file	save an object to permanent storage
ios_Sget	read data from a stream
ios_Sget_dir_lc	get the current working or naming directory
ios_Sget_ec	get a pointer to a stream eventcount
ios_Sget_handle	convert a stream ID to a handle
ios_Sinq_byte_pos	return the byte position of a stream marker
ios_Sinq_conn_flags	return connection attributes
ios_Sinq_cur_rec_len	return the length of the next record
ios_Sinq_file_attr	get file attributes
ios_Sinq_full_key	return a full seek key
ios_Sinq_mgr_flags	return manager attributes
ios_Sinq_obj_flags	return object attributes
ios_Sinq_path_name_lc	get an object's pathname
ios_Sinq_rec_pos	return the record position of a stream marker
ios_Sinq_rec_remainder	return the number of bytes left in a record
ios_Sinq_rec_type	return the record type of an object
ios_Sinq_short_key	return a short seek key
ios_Sinq_type_uid	get the type of an object
ios_Slocate	get a pointer to data in a stream
ios_Sopen	open an existing object
ios_Sput	write to an object
ios_Sreplicate	replicate a stream
ios_Sseek	seek on a stream
ios_Sseek_full_key	seek with a full key
ios_Sseek_short_key	seek with a short key
ios_Sseek_to_hof	seek to the beginning of an object
ios_Sseek_to_eof	seek to the end of an object
ios_Sset_conn_flag	set connection attributes

ios_\$set_dir .....	change the current working or naming directory
ios_\$set_locate_buffer_size .....	set the locate buffer size
ios_\$set_obj_flag .....	set object attributes
ios_\$set_rec_type .....	change an object's record type
ios_\$switch .....	switch a stream to another ID
ios_\$truncate .....	truncate an object at the current stream marker

---

## ios\_dir

intro .....	stream access to directories
ios_dir_\$isa .....	test whether a stream is open on a directory
ios_dir_\$open .....	open an object in an open directory
ios_dir_\$readdir .....	read a directory
ios_dir_\$rewinddir .....	rewind a directory
ios_dir_\$seekdir .....	seek in a directory
ios_dir_\$telldir .....	return a seek key for a directory

---

## ipc

intro .....	Domain/OS Interprocess Communication
ipc_\$close .....	close an IPC socket
ipc_\$create .....	create an IPC socket
ipc_\$delete .....	delete an IPC socket
ipc_\$get_ec .....	get an IPC eventcount
ipc_\$open .....	open an IPC socket
ipc_\$rcv .....	retrieve an IPC datagram
ipc_\$resolve .....	get an IPC socket handle
ipc_\$sar .....	send an IPC datagram and await a reply
ipc_\$send .....	send an IPC datagram
ipc_\$wait .....	wait for an IPC datagram

---

## lib

intro .....	calls to manipulate predefined large sets
-------------	---

---

## loader

intro .....	the Domain/OS loader
loader_\$clear_inlib .....	clear the installed library list
loader_\$cond_load .....	ensure a module is loaded
loader_\$inlib .....	create a privately installed library
loader_\$inquire_image .....	get image information
loader_\$inquire_known_lib .....	get information on known libraries

loader_\$inquire_section .....	get information on an image section
loader_\$inquire_stat .....	get a load history summation
loader_\$kg_define .....	add to the Known Global Table (KGT)
loader_\$kg_lookup .....	look up a symbol
loader_\$load .....	load a module
loader_\$lookup_stack_size .....	get the stack size requirement of an image
loader_\$lookup_start_addr .....	get the start address of an image

---

## mbx

intro .....	using mailboxes
mbx_\$client_window .....	return client buffer size
mbx_\$close .....	close a channel or mailbox
mbx_\$cond_get_rec_chan .....	read a channel without waiting
mbx_\$cond_get_rec_chan_set .....	read a set of channels without waiting
mbx_\$create_server .....	create a mailbox
mbx_\$deallocate .....	deallocate a channel
mbx_\$get_conditional .....	read a message without waiting
mbx_\$get_ec .....	get a pointer to a mailbox event count
mbx_\$get_rec .....	get a message from a mailbox
mbx_\$get_rec_chan .....	read a mailbox channel
mbx_\$get_rec_chan_set .....	read a set of channels
mbx_\$open .....	open a client channel in a mailbox
mbx_\$put_chr .....	send a partial message
mbx_\$put_chr_cond .....	send a partial message without waiting
mbx_\$put_rec .....	send a message
mbx_\$put_rec_cond .....	send a message without waiting
mbx_\$server_window .....	return server buffer size
mbx_\$start_helper .....	start a mailbox helper
mbx_\$timed_open .....	open a client channel with a timeout

---

## ms

intro .....	mapping file system objects
ms_\$addmap .....	map more of a object
ms_\$advise .....	advise the system on mapped object access
ms_\$attributes .....	get the file attributes of a mapped object
ms_\$crmapl .....	create, map, and lock an object
ms_\$scrip .....	create, map, and lock a temporary object
ms_\$fw_file .....	synchronize a mapped object with the file system
ms_\$fw_partial .....	partially synchronize a mapped object with the file system
ms_\$mapl .....	map and lock an object
ms_\$mapl_stream .....	permanently map an object via its XOID
ms_\$mk_permanent .....	make a temporary object permanent
ms_\$mk_temporary .....	make a permanent object temporary
ms_\$neighbors .....	find out if two mapped objects are on the same volume
ms_\$relock .....	change the lock on an object

ms_Sremap .....	map a different portion of a mapped object
ms_Struncate .....	truncate a mapped object
ms_Sunmap .....	unmap an object

---

## mts

intro .....	using magtape descriptor files
mts_\$close_desc .....	close a magtape descriptor file
mts_\$copy_desc .....	copy a magtape descriptor file
mts_\$create_default_desc .....	create a default magtape descriptor file
mts_\$get_attr .....	get a magtape descriptor attribute
mts_\$label .....	label a magtape
mts_\$open_desc .....	open a magtape descriptor file
mts_\$open_desc_uid .....	open a magtape descriptor file
mts_\$set_attr .....	set a magtape descriptor attribute

---

## mutex

intro .....	mutually exclusive resource locking
mutex_Sinit .....	initialize a mutual exclusion lock record
mutex_Slock .....	obtain a mutual exclusion lock
mutex_Sunlock .....	release a mutual exclusion lock

---

## name

intro .....	the Naming Server
name_\$add_link .....	create a link
name_\$name .....	change a leaf name
name_\$create_directory .....	create a directory
name_\$create_file .....	create a permanent file
name_\$delete_directory .....	delete a directory
name_\$delete_file .....	delete a file
name_\$drop_link .....	delete a link
name_\$extract_data_lc .....	extract a directory entry
name_\$get_ndir_lc .....	get the naming directory
name_\$get_path_lc .....	get a full pathname
name_\$get_wdir_lc .....	get the working directory
name_\$read_dir_lc .....	read a directory
name_\$read_link_lc .....	return the pathname associated with a link
name_\$set_ndir .....	set the naming directory
name_\$set_wdir .....	sets the working directory

---

## osinfo

intro .....	Domain/OS information
osinfo_\$get_rev .....	get Domain/OS revision information

---

## pad

intro .....	the Display Manager
pad_Sclear_frame .....	clear a frame
pad_Sclose_frame .....	close a frame
pad_Scooked .....	put a transcript pad in cooked mode
pad_Scpr_enable .....	enable cursor position reports
pad_Screate .....	create a new pad and window pane
pad_Screate_frame .....	create a frame
pad_Screate_icon .....	create a new pad and associated window icon
pad_Screate_window .....	create a new pad and a window to view it
pad_Sdef_pfk .....	define a program function key
pad_Sdelete_frame .....	clear and delete a frame
pad_Sdm_cmd .....	execute a Display Manager command
pad_Sedit_wait .....	wait until an edit window closes
pad_Sicon_wait .....	wait until an icon expands or moves
pad_Sinq_disp_type .....	find out the display type
pad_Sinq_font .....	find out about the current font
pad_Sinq_full_window .....	get the total display region occupied by a window
pad_Sinq_icon .....	get icon information
pad_Sinq_icon_font .....	get the current icon font
pad_Sinq_kbd .....	find out about the keyboard
pad_Sinq_position .....	find the output cursor
pad_Sinq_view .....	find out the pad region under the window
pad_Sinq_windows .....	list the windows on a pad
pad_Sis_icon .....	find out whether a window is an icon
pad_Sisa .....	test a stream for the pad trait
pad_Sisa_dm_pad .....	test whether a stream is open on a local DM pad
pad_Sload_font .....	load a character font
pad_Slocate .....	locate the keyboard cursor
pad_Smake_icon .....	change a window into an icon
pad_Smake_invisible .....	make a window invisible
pad_Smove .....	move the output cursor
pad_Spop_push_window .....	pop or push a window
pad_Sraw .....	place a pad in raw mode
pad_Sselect_window .....	make a window visible and full-size
pad_Sset_auto_close .....	close a window when its pad closes
pad_Sset_border .....	add or remove a window border
pad_Sset_full_window .....	establish a window region
pad_Sset_icon_font .....	choose a new icon font

pad\_\$set\_icon\_pos ..... establish an icon position and icon character  
 pad\_\$set\_scale ..... set a scale factor for cursor operations  
 pad\_\$set\_tabs ..... set tab stops within a pad  
 pad\_\$set\_view ..... define the pad region under a window  
 pad\_\$use\_font ..... use a new loaded font

---

## pbufs

intro ..... using paste buffers  
 pbufs\_\$create ..... create a paste buffer  
 pbufs\_\$open ..... open an existing paste buffer

---

## pfm

intro ..... managing faults  
 pfm\_\$cleanup ..... establish a clean-up handler  
 pfm\_\$enable ..... enable asynchronous faults  
 pfm\_\$error\_trap ..... simulate a fault and save a traceback  
 pfm\_\$establish\_fault\_handler ..... establish a fault handler  
 pfm\_\$inhibit ..... inhibit asynchronous faults  
 pfm\_\$release\_fault\_handler ..... release a fault handler  
 pfm\_\$reset\_cleanup ..... reset a clean-up handler  
 pfm\_\$rls\_cleanup ..... release clean-up handlers  
 pfm\_\$signal ..... signal the calling process

---

## pgm

intro ..... the program manager  
 pgm\_\$del\_arg ..... delete a program argument  
 pgm\_\$exit ..... exit a program  
 pgm\_\$get\_arg ..... get a program argument  
 pgm\_\$get\_args ..... get the program argument vector  
 pgm\_\$get\_ec ..... get a process eventcount  
 pgm\_\$get\_puid ..... get a process UID  
 pgm\_\$invoke ..... invoke a program  
 pgm\_\$make\_orphan ..... orphan a process  
 pgm\_\$proc\_wait ..... wait on a process  
 pgm\_\$set\_severity ..... set the exit severity level

---

## pm

intro ..... the Domain/OS Process Manager  
 pm\_\$get\_home\_txt ..... get the home directory

pm\_\$get\_sid\_txt ..... get the SID

---

## prf

intro ..... the Domain/OS print library  
prf\_\$config\_file ..... set print options from a file  
prf\_\$edit\_job ..... edit a print job at the current site  
prf\_\$get\_printers ..... get a list of printers at the current site  
prf\_\$get\_sites ..... get a list of print sites  
prf\_\$init ..... initialize print request options  
prf\_\$inq\_option ..... get a print request option  
prf\_\$name\_print ..... spool a file for printing  
prf\_\$queue\_file ..... queue a print request  
prf\_\$read\_queue ..... get a list of print jobs in the queue  
prf\_\$set\_option ..... set an option in a print request  
prf\_\$signal\_printer ..... signal a print job  
prf\_\$stream\_print ..... print from a stream

---

## procl

intro ..... the Level 1 Process Manager  
procl\_\$get\_cput ..... get elapsed CPU time

---

## proc2

intro ..... the Level 2 Process Manager  
proc2\_\$get\_info ..... get level 2 process information  
proc2\_\$list ..... list level 2 process UIDs  
proc2\_\$who\_am\_i ..... get the UID of the calling process

---

## rws

intro ..... dynamic storage allocation  
rws\_\$alloc ..... allocate storage  
rws\_\$alloc\_heap\_pool ..... allocate heap storage  
rws\_\$alloc\_rw\_pool ..... allocate read/write storage from a pool  
rws\_\$release\_heap\_pool ..... release heap storage

---

## sio

intro ..... controlling serial I/O lines

sio\_\$control ..... set serial line options  
sio\_\$inquire ..... get serial line options

---

## status

intro ..... status reporting types and constants

---

## task

intro ..... the Domain/OS task library  
task\_\$blast ..... kill a task without cleanup  
task\_\$create ..... create a task  
task\_\$exit ..... exit a task  
task\_\$get\_ec ..... get a completion eventcount  
task\_\$get\_handle ..... return the task handle  
task\_\$get\_info ..... get information about a task  
task\_\$release ..... release a task and report  
task\_\$set\_name ..... name a task  
task\_\$set\_result ..... change the completion status and output value  
task\_\$signal ..... signal a task  
task\_\$tasking\_enabled ..... determine whether tasking is enabled  
task\_\$yield ..... yield the processor

---

## time

intro ..... the Domain/OS time service  
time\_\$clock ..... get the system clock value  
time\_\$get\_ec ..... get a pointer to the time eventcount  
time\_\$wait ..... wait for an interval

---

## tone

intro ..... make a noise  
tone\_\$time ..... make a noise of a specified duration

---

## tpad

intro ..... locator (touchpad) manager calls  
tpad\_\$inq\_dtype ..... return the last locator used  
tpad\_\$inquire ..... get information about the current locator response  
tpad\_\$re\_range ..... re-establishes the touchpad raw data range

tpad\_\$set\_cursor ..... re-establish the locator origin in relative mode  
 tpad\_\$set\_mode ..... set pointing device response characteristics

---

## vec

intro ..... the Vector Library  
 vec\_\$add\_constant ..... add a scalar to a single-precision vector  
 vec\_\$add\_constant\_i ..... add a scalar to a vector in a single-precision matrix  
 vec\_\$add\_vector ..... add two single-precision vectors  
 vec\_\$add\_vector\_i ..... add vectors in two single-precision matrixes  
 vec\_\$copy ..... copy a single-precision vector  
 vec\_\$copy\_i ..... copy a vector from one single-precision matrix to another  
 vec\_\$dadd\_constant ..... add a scalar to a double-precision vector  
 vec\_\$dadd\_constant\_i ..... add a scalar to a vector in a double-precision matrix  
 vec\_\$dadd\_vector ..... add two double-precision vectors  
 vec\_\$dadd\_vector\_i ..... add vectors in two double-precision matrixes  
 vec\_\$dcopy ..... copy a double-precision vector  
 vec\_\$dcopy\_i ..... copy a vector from one double-precision matrix to another  
 vec\_\$ddot ..... return the dot product of two double-precision vectors  
 vec\_\$ddot\_i ..... return the dot product of two vectors in double-precision matrixes  
 vec\_\$dinit ..... initialize a double-precision vector  
 vec\_\$dmat\_mult ..... multiply two 4x4 double-precision matrixes  
 vec\_\$dmat\_multn ..... multiply two double-precision matrixes  
 vec\_\$dmax ..... find the maximum absolute value in a double-precision vector  
 vec\_\$dmax\_i ..... find the maximum absolute value in a vector from a double-precision matrix  
 vec\_\$dmult\_add ..... scale and add one double-precision vector to another  
 vec\_\$dmult\_add\_i ..... scale and add double-precision vectors in matrixes  
 vec\_\$dmult\_constant ..... multiply a double-precision vector by a scalar  
 vec\_\$dmult\_constant\_i ..... multiply a vector in a double-precision matrix by a scalar  
 vec\_\$dot ..... return the dot product of two single-precision vectors  
 vec\_\$dot\_i ..... return the dot product of two vectors in single-precision matrixes  
 vec\_\$dp\_sp ..... copy a double-precision vector to a single-precision vector  
 vec\_\$dp\_sp\_i ..... copy a vector from a double-precision matrix into a single-precision matrix  
 vec\_\$dpostmult ..... multiply a double-precision vector by a 4x4 matrix  
 vec\_\$dpostmultn ..... multiply a double-precision vector by a matrix  
 vec\_\$dpremult ..... multiply a double-precision vector by a 4x4 matrix  
 vec\_\$dpremultn ..... multiply a double-precision vector by a matrix  
 vec\_\$dsub ..... subtract double-precision vectors  
 vec\_\$dsub\_i ..... subtract double-precision vectors in matrixes  
 vec\_\$dsum ..... sum the elements of a double-precision vector  
 vec\_\$dsum\_i ..... sum the elements of a vector in a double-precision matrix  
 vec\_\$dswap ..... swap two double-precision vectors  
 vec\_\$dswap\_i ..... swap two vectors in a double-precision matrix  
 vec\_\$dzero ..... zero a double-precision vector  
 vec\_\$dzero\_i ..... zero a vector in a double-precision matrix  
 vec\_\$iadd\_constant ..... add a scalar constant to a 32-bit integer vector  
 vec\_\$iadd\_constant16 ..... add a scalar constant to a 16-bit integer vector  
 vec\_\$iadd\_constant16\_i ..... add a scalar to a vector in a 16-bit integer matrix  
 vec\_\$iadd\_constant\_i ..... add a scalar to a vector in a 32-bit integer matrix

vec_siadd_vector .....	add two 32-bit integer vectors
vec_siadd_vector16 .....	add two 16-bit integer vectors
vec_siadd_vector16_i .....	add vectors in two 16-bit integer matrixes
vec_siadd_vector_i .....	add vectors in two 32-bit integer matrixes
vec_sicopy .....	copy a 32-bit integer vector
vec_sicopy16 .....	copy a 16-bit integer vector
vec_sicopy16_i .....	copy a vector from one 16-bit integer matrix to another
vec_sicopy_i .....	copy a vector from one 32-bit integer matrix to another
vec_sidot .....	return the dot product of two 32-bit integer vectors
vec_sidot16 .....	return the dot product of two 16-bit integer vectors
vec_sidot16_i .....	return the dot product of two vectors in 16-bit integer matrixes
vec_sidot_i .....	return the dot product of two vectors in 32-bit integer matrixes
vec_siinit .....	initialize a 32-bit integer vector
vec_siinit16 .....	initialize a 16-bit integer vector
vec_simat_mult .....	multiply two 4x4 32-bit integer matrixes
vec_simat_mult16 .....	multiply two 4x4 16-bit integer matrixes
vec_simat_multn .....	multiply two 32-bit integer matrixes
vec_simat_multn16 .....	multiply two 16-bit integer matrixes
vec_simax .....	find the maximum absolute value in a 32-bit integer vector
vec_simax16 .....	find the maximum absolute value in a 16-bit integer vector
vec_simax16_i .....	find the maximum absolute value in a vector from a 16-bit integer matrix
vec_simax_i .....	find the maximum absolute value in a vector from a 32-bit integer matrix
vec_simult_add .....	scale and add one 32-bit integer vector to another
vec_simult_add16 .....	scale and add one 16-bit integer vector to another
vec_simult_add16_i .....	scale and add 16-bit vectors in matrixes
vec_simult_add_i .....	scale and add 32-bit vectors in matrixes
vec_simult_constant .....	multiply a 32-bit integer vector by a scalar
vec_simult_constant16 .....	multiply a 16-bit integer vector by a scalar
vec_simult_constant16_i .....	multiply a vector in a 16-bit integer matrix by a scalar
vec_simult_constant_i .....	multiply a vector in a 32-bit integer matrix by a scalar
vec_sinit .....	initialize a single-precision vector
vec_sipostmult .....	multiply a 32-bit integer vector by a 4x4 matrix
vec_sipostmult16 .....	multiply a 16-bit integer vector by a 4x4 matrix
vec_sipostmultn .....	multiply a 32-bit integer vector by a matrix
vec_sipostmultn16 .....	multiply a 16-bit integer vector by a matrix
vec_sipremult .....	multiply a 32-bit vector by a 4x4 matrix
vec_sipremult16 .....	multiply a 16-bit vector by a 4x4 matrix
vec_sipremultn .....	multiply a 32-bit integer vector by a matrix
vec_sipremultn16 .....	multiply a 16-bit integer vector by a matrix
vec_sisub .....	subtract 32-bit integer vectors
vec_sisub16 .....	subtract 16-bit integer vectors
vec_sisub16_i .....	subtract 16-bit integer vectors in matrixes
vec_sisub_i .....	subtract 32-bit integer vectors in matrixes
vec_sisum .....	sum the elements of a 32-bit integer vector
vec_sisum16 .....	sum the elements of a 16-bit integer vector
vec_sisum16_i .....	sum the elements of a vector in a 16-bit integer matrix
vec_sisum_i .....	sum the elements of a vector in a 32-bit integer matrix
vec_siswap .....	swap two 32-bit integer vectors
vec_siswap16 .....	swap two 16-bit integer vectors
vec_siswap16_i .....	swap two vectors in a 16-bit integer matrix
vec_siswap_i .....	swap two vectors in a 32-bit integer matrix

vec_\$zero	zero a 32-bit integer vector
vec_\$zero16	zero a 16-bit integer vector
vec_\$zero16_i	zero a vector in a 16-bit integer matrix
vec_\$zero_i	zero a vector in a 32-bit integer matrix
vec_\$mat_mult	multiply two 4x4 single-precision matrixes
vec_\$mat_multn	multiply two single-precision matrixes
vec_\$max	find the maximum absolute value in a single-precision vector
vec_\$max_i	find the maximum absolute value in a vector from a single-precision matrix
vec_\$mult_add	scale and add one single-precision vector to another
vec_\$mult_add_i	scale and add single-precision vectors in matrixes
vec_\$mult_constant	multiply a single-precision vector by a scalar
vec_\$mult_constant_i	multiply a vector in a single-precision matrix by a scalar
vec_\$postmult	multiply a single-precision vector by a 4x4 matrix
vec_\$postmultn	multiply a single-precision vector by a matrix
vec_\$premult	multiply a single-precision vector by a 4x4 matrix
vec_\$premultn	multiply a single-precision vector by a matrix
vec_\$sp_dp	copy a single-precision vector to a double-precision vector
vec_\$sp_dp_i	copy a single-precision vector to a double-precision vector in matrixes
vec_\$sub	subtract single-precision vectors
vec_\$sub_i	subtract single-precision vectors in matrixes
vec_\$sum	sum the elements of a single-precision vector
vec_\$sum_i	sum the elements of a vector in a single-precision matrix
vec_\$swap	swap two single-precision vectors
vec_\$swap_i	swap two vectors in a single-precision matrix
vec_\$zero	zero a single-precision vector
vec_\$zero_i	zero a vector in a single-precision matrix

---

## vfmt

intro	variable formatting package
vfmt_\$decode10	formatted read from a string
vfmt_\$decode2	formatted read from a string
vfmt_\$decode5	formatted read from a string
vfmt_\$encode	formatted write to a string
vfmt_\$encode10	formatted write to a string
vfmt_\$encode2	formatted write to a string
vfmt_\$encode5	formatted write to a string
vfmt_\$read10	formatted read from standard input
vfmt_\$read2	formatted read from standard input
vfmt_\$read5	formatted read from standard input
vfmt_\$rs10	formatted read from a stream
vfmt_\$rs2	formatted read from a stream
vfmt_\$rs5	formatted read from a stream
vfmt_\$write	formatted write to standard output
vfmt_\$write10	formatted write to standard output
vfmt_\$write2	formatted write to standard output
vfmt_\$write5	formatted write to standard output
vfmt_\$ws	formatted write to a stream
vfmt_\$ws10	formatted write to a stream

vfmt\_\$ws2 ..... formatted write to a stream  
vfmt\_\$ws5 ..... formatted write to a stream

## Permuted Index

---

scalar /multiply a vector in a	16-bit integer matrix by a	vec_\$imult_constant16_i
/copy a vector from one	16-bit integer matrix to/	vec_\$icopy16_i
/add a scalar to a vector in a	16-bit integer matrix	vec_\$iadd_constant16_i
/value in a vector from a	16-bit integer matrix	vec_\$imax16_i
the elements of a vector in a	16-bit integer matrix /sum	vec_\$isum16_i
/swap two vectors in a	16-bit integer matrix	vec_\$iswap16_i
/zero a vector in a	16-bit integer matrix	vec_\$izero16_i
/add vectors in two	16-bit integer matrixes	vec_\$iadd_vector16_i
dot product of two vectors in	16-bit integer matrixes /the	vec_\$idot16_i
/multiply two 4x4	16-bit integer matrixes	vec_\$imat_mult16
/multiply two	16-bit integer matrixes	vec_\$imat_multn16
vec_\$ipostmult16: multiply a	16-bit integer vector by a/	vec_\$ipostmult16
vec_\$ipostmultn16: multiply a	16-bit integer vector by a/	vec_\$ipostmultn16
vec_\$ipremult16: multiply a	16-bit integer vector by a/	vec_\$ipremult16
scalar /multiply a	16-bit integer vector by a	vec_\$imult_constant16
another /scale and add one	16-bit integer vector to	vec_\$imult_add16
/add a scalar constant to a	16-bit integer vector	vec_\$iadd_constant16
vec_\$icopy16: copy a	16-bit integer vector	vec_\$icopy16
vec_\$iinit16: initialize a	16-bit integer vector	vec_\$iinit16
maximum absolute value in a	16-bit integer vector /the	vec_\$imax16
/sum the elements of a	16-bit integer vector	vec_\$isum16
vec_\$izero16: zero a	16-bit integer vector	vec_\$izero16
vec_\$isub16_i: subtract	16-bit integer vectors in/	vec_\$isub16_i
vec_\$iadd_vector16: add two	16-bit integer vectors	vec_\$iadd_vector16
/return the dot product of two	16-bit integer vectors	vec_\$idot16
vec_\$isub16: subtract	16-bit integer vectors	vec_\$isub16
vec_\$iswap16: swap two	16-bit integer vectors	vec_\$iswap16
vec_\$ipremult16: multiply a	16-bit vector by a 4x4/	vec_\$ipremult16
/scale and add	16-bit vectors in matrixes	vec_\$imult_add16_i
scalar /multiply a vector in a	32-bit integer matrix by a	vec_\$imult_constant_i
/copy a vector from one	32-bit integer matrix to/	vec_\$icopy_i
/add a scalar to a vector in a	32-bit integer matrix	vec_\$iadd_constant_i
/value in a vector from a	32-bit integer matrix	vec_\$imax_i
the elements of a vector in a	32-bit integer matrix /sum	vec_\$isum_i
/swap two vectors in a	32-bit integer matrix	vec_\$iswap_i
/zero a vector in a	32-bit integer matrix	vec_\$izero_i
/add vectors in two	32-bit integer matrixes	vec_\$iadd_vector_i
dot product of two vectors in	32-bit integer matrixes /the	vec_\$idot_i

/multiply two 4x4	32-bit integer matrixes	vec_Simat_mult
vec_Simat_multn: multiply two	32-bit integer matrixes	vec_Simat_multn
vec_Sipostmult: multiply a	32-bit integer vector by a/	vec_Sipostmult
vec_Sipostmultn: multiply a	32-bit integer vector by a/	vec_Sipostmultn
vec_Sipremultn: multiply a	32-bit integer vector by a/	vec_Sipremultn
scalar /multiply a	32-bit integer vector by a	vec_Simult_constant
another /scale and add one	32-bit integer vector to	vec_Simult_add
/add a scalar constant to a	32-bit integer vector	vec_Siadd_constant
vec_Sicopy: copy a	32-bit integer vector	vec_Sicopy
vec_Siinit: initialize a	32-bit integer vector	vec_Siinit
maximum absolute value in a	32-bit integer vector /the	vec_Simax
/sum the elements of a	32-bit integer vector	vec_Sisum
vec_Sizero: zero a	32-bit integer vector	vec_Sizero
vec_Sisub_i: subtract	32-bit integer vectors in/	vec_Sisub_i
vec_Siadd_vector: add two	32-bit integer vectors	vec_Siadd_vector
/return the dot product of two	32-bit integer vectors	vec_Sidot
vec_Sisub: subtract	32-bit integer vectors	vec_Sisub
vec_Siswap: swap two	32-bit integer vectors	vec_Siswap
vec_Sipremult: multiply a	32-bit vector by a 4x4/	vec_Sipremult
/scale and add	32-bit vectors in matrixes	vec_Simult_add_i
vec_Simat_mult16: multiply two	4x4 16-bit integer matrixes	vec_Simat_mult16
vec_Simat_mult: multiply two	4x4 32-bit integer matrixes	vec_Simat_mult
vec_Sdmat_mult: multiply two	4x4 double-precision/	vec_Sdmat_mult
a double-precision vector by a	4x4 matrix /multiply	vec_Sdpostmult
a double-precision vector by a	4x4 matrix /multiply	vec_Sdpremult
a 32-bit integer vector by a	4x4 matrix /multiply	vec_Sipostmult
a 16-bit integer vector by a	4x4 matrix /multiply	vec_Sipostmult16
multiply a 32-bit vector by a	4x4 matrix vec_Sipremult:	vec_Sipremult
/multiply a 16-bit vector by a	4x4 matrix	vec_Sipremult16
a single-precision vector by a	4x4 matrix /multiply	vec_Spostmult
a single-precision vector by a	4x4 matrix /multiply	vec_Spremult
vec_Smat_mult: multiply two	4x4 single-precision/	vec_Smat_mult
vec_Simax16: find the maximum	absolute value in a 16-bit/	vec_Simax16
vec_Simax: find the maximum	absolute value in a 32-bit/	vec_Simax
vec_Sdmax: find the maximum	absolute value in a/	vec_Sdmax
vec_Smax: find the maximum	absolute value in a/	vec_Smax
from a/ /find the maximum	absolute value in a vector	vec_Simax16_i
vec_Simax_i: find the maximum	absolute value in a vector/	vec_Simax_i
vec_Sdmax_i: find the maximum	absolute value in a vector/	vec_Sdmax_i
vec_Smax_i: find the maximum	absolute value in a vector/	vec_Smax_i
the system on mapped object	access ms_Sadvice: advise	ms_Sadvice
ios_dir_Sintro: stream	access to directories	ios_dir_Sintro
manager rights	aclm_Sdown: deassert subsystem	aclm_Sdown
protected subsystem privilege/	aclm_Sintro: change the	aclm_Sintro
manager rights	aclm_Sup: assert subsystem	aclm_Sup
vec_Simult_add16_i: scale and	add 16-bit vectors in matrixes	vec_Simult_add16_i
vec_Simult_add_i: scale and	add 32-bit vectors in matrixes	vec_Simult_add_i
16-bit/ vec_Siadd_constant16:	add a scalar constant to a	vec_Siadd_constant16
32-bit/ vec_Siadd_constant:	add a scalar constant to a	vec_Siadd_constant
vec_Sdadd_constant:	add a scalar to a/	vec_Sdadd_constant
vec_Sadd_constant:	add a scalar to a/	vec_Sadd_constant

vec_Siadd_constant16_i:	add a scalar to a vector in a/	vec_Siadd_constant16_i
32-bit/ vec_Siadd_constant_i:	add a scalar to a vector in a	vec_Siadd_constant_i
vec_Sdadd_constant_i:	add a scalar to a vector in a/	vec_Sdadd_constant_i
vec_\$dadd_constant_i:	add a scalar to a vector in a/	vec_\$dadd_constant_i
vec_Sdmult_add_i:	scale and add double-precision vectors/	vec_Sdmult_add_i
vec_Simult_add16:	scale and add one 16-bit integer vector/	vec_Simult_add16
to/ vec_Simult_add:	scale and add one 32-bit integer vector	vec_Simult_add
vec_Sdmult_add:	scale and add one double-precision/	vec_Sdmult_add
vec_Smult_add:	scale and add one single-precision/	vec_Smult_add
pad_Sset_border:	add or remove a window border	pad_Sset_border
m/ vec_Smult_add_i:	scale and add single-precision vectors	vec_Smult_add_i
(KGT) loader_Skg_define:	add to the Known Global Table	loader_Skg_define
vec_Siadd_vector16:	add two 16-bit integer vectors	vec_Siadd_vector16
vec_Siadd_vector:	add two 32-bit integer vectors	vec_Siadd_vector
cal_Sadd_clock:	add two clock values	cal_Sadd_clock
vectors vec_Sdadd_vector:	add two double-precision	vec_Sdadd_vector
vectors vec_Sadd_vector:	add two single-precision	vec_Sadd_vector
integer/ vec_Siadd_vector16_i:	add vectors in two 16-bit	vec_Siadd_vector16_i
integer/ vec_Siadd_vector_i:	add vectors in two 32-bit	vec_Siadd_vector_i
vec_Sdadd_vector_i:	add vectors in two/	vec_Sdadd_vector_i
vec_Sadd_vector_i:	add vectors in two/	vec_Sadd_vector_i
/get the start	address of an image	loader_Slookup_start_addr
object access ms_Sadvise:	advise the system on mapped	ms_Sadvise
rws_Salloc_heap_pool:	allocate heap storage	rws_Salloc_heap_pool
from a/ rws_Salloc_rw_pool:	allocate read/write storage	rws_Salloc_rw_pool
rws_Salloc:	allocate storage	rws_Salloc
ctm_Salloc_pv:	allocate unused pixel values	ctm_Salloc_pv
rws_Sintro:	dynamic storage allocation	rws_Sintro
pgm_Sdel_arg:	delete a program argument	pgm_Sdel_arg
pgm_Sget_arg:	get a program argument	pgm_Sget_arg
pgm_Sget_args:	get the program argument vector	pgm_Sget_args
/decode an	ASCII date	cal_Sdecode_ascii_date
/decode an	ascii time	cal_Sdecode_ascii_time
/decode an	ASCII time zone	cal_Sdecode_ascii_tzdiff
rights aclm_Sup:	assert subsystem manager	aclm_Sup
/create a new pad and	associated window icon	pad_Screate_icon
/return the pathname	associated with a link	name_Sread_link_lc
for an eventcount trigger or	asynchronous fault /wait	ec2_Swait_svc
pfm_Senable:	enable asynchronous faults	pfm_Senable
pfm_Sinhibit:	inhibit asynchronous faults	pfm_Sinhibit
get a magtape descriptor	attribute mts_Sget_attr:	mts_Sget_attr
set a magtape descriptor	attribute mts_Sset_attr:	mts_Sset_attr
/return connection	attributes	ios_Sinq_conn_flags
ios_Sinq_file_attr:	get file attributes	ios_Sinq_file_attr
/return manager	attributes	ios_Sinq_mgr_flags
/return object	attributes	ios_Sinq_obj_flags
/set connection	attributes	ios_Sset_conn_flag
ios_Sset_obj_flag:	set object attributes	ios_Sset_obj_flag
ms_Sattributes:	get the file attributes of a mapped object	ms_Sattributes
send an IPC datagram and	await a reply ipc_Ssar:	ipc_Ssar
ios_Sseek_to_bof:	seek to the beginning of an object	ios_Sseek_to_bof

error_\$\$fail: test the fail	bit in a completion status .....	error_\$\$fail
gmf_\$\$restore_plane: restore a	bitmap from a GMF .....	gmf_\$\$restore_plane
gmf_\$\$copy_plane: copy a	bitmap plane to a GMF .....	gmf_\$\$copy_plane
/copy part of a	bitmap plane to a GMF .....	gmf_\$\$copy_subplane
add or remove a window	border pad_\$\$set_border: .....	pad_\$\$set_border
pbufs_\$\$create: create a paste	buffer .....	pbufs_\$\$create
open an existing paste	buffer pbufs_\$\$open: .....	pbufs_\$\$open
/set the locate	buffer size .....	ios_\$\$set_locate_buffer_size
/return client	buffer size .....	mbx_\$\$client_window
/return server	buffer size .....	mbx_\$\$server_window
pbufs_\$\$intro: using paste	buffers .....	pbufs_\$\$intro
ios_\$\$inq_byte_pos: return the	byte position of a stream/ .....	ios_\$\$inq_byte_pos
/return the number of	bytes left in a record .....	ios_\$\$inq_rec_remainder
values	cal_\$\$add_clock: add two clock .....	cal_\$\$add_clock
convert a UTC clock value to/	cal_\$\$supply_local_offset: .....	cal_\$\$supply_local_offset
clock ticks into whole/	cal_\$\$s_clock_to_sec: convert .....	cal_\$\$s_clock_to_sec
clock values	cal_\$\$scmp_clock: compare two .....	cal_\$\$scmp_clock
an ASCII date	cal_\$\$sdecode_ascii_date: decode .....	cal_\$\$sdecode_ascii_date
an ascii time	cal_\$\$sdecode_ascii_time: decode .....	cal_\$\$sdecode_ascii_time
decode an ASCII time zone	cal_\$\$sdecode_ascii_tzdif: .....	cal_\$\$sdecode_ascii_tzdif
the local date and time	cal_\$\$sdecode_local_time: get .....	cal_\$\$sdecode_local_time
clock value	cal_\$\$sdecode_time: decode a .....	cal_\$\$sdecode_time
date and time	cal_\$\$sencode_time: encode a .....	cal_\$\$sencode_time
cal_\$\$sintro: the Domain/OS	calendar service .....	cal_\$\$sintro
clock ticks into seconds	cal_\$\$sfloat_clock: convert .....	cal_\$\$sfloat_clock
time zone	cal_\$\$sget_info: get the local .....	cal_\$\$sget_info
local time	cal_\$\$sget_local_time: get the .....	cal_\$\$sget_local_time
service	cal_\$\$sintro: the Domain/OS calendar .....	cal_\$\$sintro
pfm_\$\$signal: signal the	calling process .....	pfm_\$\$signal
/get the UID of the	calling process .....	proc2_\$\$who_am_i
large sets lib_\$\$sintro:	calls to manipulate predefined .....	lib_\$\$sintro
locator (touchpad) manager	calls tpad_\$\$sintro: .....	tpad_\$\$sintro
convert local time to UTC	cal_\$\$sremove_local_offset: .....	cal_\$\$sremove_local_offset
seconds to a clock value	cal_\$\$ssec_to_clock: convert .....	cal_\$\$ssec_to_clock
clock values	cal_\$\$s_sub_clock: subtract two .....	cal_\$\$s_sub_clock
weekday for a date	cal_\$\$sweekday: return the .....	cal_\$\$sweekday
local time zone	cal_\$\$swrite_timezone: set the .....	cal_\$\$swrite_timezone
mbx_\$\$sopen: open a client	channel in a mailbox .....	mbx_\$\$sopen
mbx_\$\$sdeallocate: deallocate a	channel .....	mbx_\$\$sdeallocate
read a mailbox	channel mbx_\$\$sget_rec_chan: .....	mbx_\$\$sget_rec_chan
mbx_\$\$sclose: close a	channel or mailbox .....	mbx_\$\$sclose
mbx_\$\$stimed_open: open a client	channel with a timeout .....	mbx_\$\$stimed_open
mbx_\$\$scond_get_rec_chan: read a	channel without waiting .....	mbx_\$\$scond_get_rec_chan
/read a set of	channels .....	mbx_\$\$sget_rec_chan_set
/read a set of	channels without waiting .....	mbx_\$\$scond_get_rec_chan_set
pad_\$\$sload_font: load a	character font .....	pad_\$\$sload_font
an icon position and icon	character /establish .....	pad_\$\$sset_icon_pos
/set pointing device response	characteristics .....	tpad_\$\$sset_mode
pad_\$\$sset_icon_font:	choose a new icon font .....	pad_\$\$sset_icon_font
pfm_\$\$scleanup: establish a	clean-up handler .....	pfm_\$\$scleanup
pfm_\$\$sreset_cleanup: reset a	clean-up handler .....	pfm_\$\$sreset_cleanup

pfm_Srls_cleanup: release	clean-up handlers .....	pfm_Srls_cleanup
kill a task without	cleanup task_Sblast: .....	task_Sblast
pad_Sclear_frame:	clear a frame .....	pad_Sclear_frame
pad_Sdelete_frame:	clear and delete a frame .....	pad_Sdelete_frame
list_loader_Sclear_inlib:	clear the installed library .....	loader_Sclear_inlib
mbx_Sclient_window: return	client buffer size .....	mbx_Sclient_window
mbx_Sopen: open a	client channel in a mailbox .....	mbx_Sopen
mbx_Stimed_open: open a	client channel with a timeout .....	mbx_Stimed_open
cal_Sfloat_clock: convert	clock ticks into seconds .....	cal_Sfloat_clock
cal_Slock_to_sec: convert	clock ticks into whole seconds .....	cal_Slock_to_sec
cal_Sdecode_time: decode a	clock value .....	cal_Sdecode_time
/convert seconds to a	clock value .....	cal_Ssec_to_clock
time_Sclock: get the system	clock value .....	time_Sclock
/convert a UTC	clock value to a local one .....	cal_Sapply_local_offset
cal_Sadd_clock: add two	clock values .....	cal_Sadd_clock
cal_Scmp_clock: compare two	clock values .....	cal_Scmp_clock
cal_Ssub_clock: subtract two	clock values .....	cal_Ssub_clock
mbx_Sclose:	close a channel or mailbox .....	mbx_Sclose
pad_Sclose_frame:	close a frame .....	pad_Sclose_frame
gmf_Sclose:	close a GMF .....	gmf_Sclose
file_mts_Sclose_desc:	close a magtape descriptor .....	mts_Sclose_desc
ios_Sclose:	close a stream .....	ios_Sclose
closes pad_Sset_auto_close:	close a window when its pad .....	pad_Sset_auto_close
ipc_Sclose:	close an IPC socket .....	ipc_Sclose
wait until an edit window	closes pad_Sedit_wait: .....	pad_Sedit_wait
close a window when its pad	closes pad_Sset_auto_close: .....	pad_Sset_auto_close
error_Scode: return the error	code from a completion status .....	error_Scode
/return the module	code from a completion status .....	error_Smodule
/return the subsystem	code from a completion status .....	error_Ssubsys
fault_Sintro: Domain/OS fault	codes .....	fault_Sintro
ctm_Sfind_color: find a	color in the color map .....	ctm_Sfind_color
find a color in the	color map ctm_Sfind_color: .....	ctm_Sfind_color
ctm_Sintro:	Color Table Manager .....	ctm_Sintro
execute a Display Manager	command pad_Sdm_cmd: .....	pad_Sdm_cmd
ipc_Sintro: Domain/OS Interprocess	Communication .....	ipc_Sintro
cal_Scmp_clock:	compare two clock values .....	cal_Scmp_clock
task_Sget_ec: get a	completion eventcount .....	task_Sget_ec
task_Sset_result: change the	completion status and output/ .....	task_Sset_result
/return the error code from a	completion status .....	error_Scode
/test the fail bit in a	completion status .....	error_Sfail
/return the module code from a	completion status .....	error_Smodule
the subsystem code from a	completion status /return .....	error_Ssubsys
ios_Sinq_conn_flags: return	connection attributes .....	ios_Sinq_conn_flags
ios_Sset_conn_flag: set	connection attributes .....	ios_Sset_conn_flag
vector /add a scalar	constant to a 16-bit integer .....	vec_Siadd_constant16
vector /add a scalar	constant to a 32-bit integer .....	vec_Siadd_constant
status reporting types and	constants status_Sintro: .....	status_Sintro
/modify the floating-point	control register .....	fpp_Scontrol
sio_Sintro:	controlling serial I/O lines .....	sio_Sintro
handle ios_Sget_handle:	convert a stream ID to a .....	ios_Sget_handle
cal_Sapply_local_offset:	convert a UTC clock value to a/ .....	cal_Sapply_local_offset

seconds	cal_\$float_clock:	convert clock ticks into .....	cal_\$float_clock
seconds	cal_\$lock_to_sec:	convert clock ticks into whole .....	cal_\$lock_to_sec
cal_\$remove_local_offset:		convert local time to UTC .....	cal_\$remove_local_offset
value	cal_\$sec_to_clock:	convert seconds to a clock .....	cal_\$sec_to_clock
put a transcript pad in	cooked mode	pad_\$cooked:	pad_\$cooked
	vec_\$icopy16:	copy a 16-bit integer vector .....	vec_\$icopy16
	vec_\$icopy:	copy a 32-bit integer vector .....	vec_\$icopy
	gmf_\$copy_plane:	copy a bitmap plane to a GMF .....	gmf_\$copy_plane
to a/	vec_\$dp_sp:	copy a double-precision vector .....	vec_\$dp_sp
	vec_\$dcopy:	copy a double-precision vector .....	vec_\$dcopy
mts_\$copy_desc:		copy a magtape descriptor file .....	mts_\$copy_desc
to a/	vec_\$sp_dp_i:	copy a single-precision vector .....	vec_\$sp_dp_i
to a/	vec_\$sp_dp:	copy a single-precision vector .....	vec_\$sp_dp
	vec_\$scopy:	copy a single-precision vector .....	vec_\$scopy
	vec_\$dp_sp_i:	copy a vector from a/ .....	vec_\$dp_sp_i
integer/	vec_\$copy16_i:	copy a vector from one 16-bit .....	vec_\$copy16_i
integer matrix/	vec_\$copy_i:	copy a vector from one 32-bit .....	vec_\$copy_i
	vec_\$dcopy_i:	copy a vector from one/ .....	vec_\$dcopy_i
single-precision/	vec_\$copy_i:	copy a vector from one .....	vec_\$copy_i
a GMF	gmf_\$copy_subplane:	copy part of a bitmap plane to .....	gmf_\$copy_subplane
increment pixel value use	counts	ctm_\$inc_use_count:	ctm_\$inc_use_count
decrement pixel value use	counts	ctm_\$release_pv:	ctm_\$release_pv
procl_\$get_cput:	get elapsed	CPU time .....	procl_\$get_cput
mts_\$create_default_desc:	create a default magtape/ .....		mts_\$create_default_desc
name	_\$create_directory:	create a directory .....	name_\$create_directory
	pad_\$create_frame:	create a frame .....	pad_\$create_frame
	name	_\$add_link:	create a link .....
	mbx_\$create_server:	create a mailbox .....	mbx_\$create_server
to view/	pad_\$create_window:	create a new pad and a window .....	pad_\$create_window
associated/	pad_\$create_icon:	create a new pad and .....	pad_\$create_icon
	pane	pad_\$create:	create a new pad and window .....
	pbufs	_\$create:	create a paste buffer .....
	name	_\$create_file:	create a permanent file .....
	library loader	_\$inlib:	create a privately installed .....
	task	_\$create:	create a task .....
	ipc	_\$create:	create an IPC socket .....
stream to it	ios	_\$create:	create an object and open a .....
temporary object	ms_\$cretemp:	create, map, and lock a .....	ms_\$cretemp
	object	ms_\$cremapl:	create, map, and lock an .....
	pixel values	ctm_\$alloc_pv:	allocate unused .....
	in the color map	ctm_\$find_color:	find a color .....
pixel value use counts	ctm_\$inc_use_count:	increment .....	ctm_\$inc_use_count
Manager	ctm_\$intro:	Color Table .....	ctm_\$intro
pixel values with other/	ctm_\$mark_read_only:	share .....	ctm_\$mark_read_only
pixel value use counts	ctm_\$release_pv:	decrement .....	ctm_\$release_pv
fpp_\$save_restore:	save the	current and load a new/ .....	fpp_\$save_restore
	find out about the	current font	pad_\$inq_font:
	/get information about the	current icon font .....	pad_\$inq_icon_font
	edit a print job at the	current locator response .....	tpad_\$inquire
	/get a list of printers at the	current site	prf_\$edit_job:
		current site .....	prf_\$get_printers

/truncate an object at the	current stream marker .....	ios_\$truncate
ios_\$get_dir_lc: get the	current working or naming/ .....	ios_\$get_dir_lc
ios_\$set_dir: change the	current working or naming/ .....	ios_\$set_dir
/set a scale factor for	cursor operations .....	pad_\$set_scale
find the output	cursor pad_\$inq_position: .....	pad_\$inq_position
locate the keyboard	cursor pad_\$locate: .....	pad_\$locate
pad_\$move: move the output	cursor .....	pad_\$move
pad_\$cpr_enable: enable	cursor position reports .....	pad_\$cpr_enable
ios_\$get: read	data from a stream .....	ios_\$get
ios_\$locate: get a pointer to	data in a stream .....	ios_\$locate
the touchpad raw	data range /re-establishes .....	tpad_\$re_range
ipc_\$sar: send an IPC	datagram and await a reply .....	ipc_\$sar
ipc_\$rcv: retrieve an IPC	datagram .....	ipc_\$rcv
ipc_\$send: send an IPC	datagram .....	ipc_\$send
ipc_\$wait: wait for an IPC	datagram .....	ipc_\$wait
/get the local	date and time .....	cal_\$decode_local_time
cal_\$encode_time: encode a	date and time .....	cal_\$encode_time
decode an ASCII	date cal_\$decode_ascii_date: .....	cal_\$decode_ascii_date
return the weekday for a	date cal_\$weekday: .....	cal_\$weekday
mbx_\$deallocate: deallocate a channel	.....	mbx_\$deallocate
rights aclm_\$down:	deassert subsystem manager .....	aclm_\$down
cal_\$decode_time:	decode a clock value .....	cal_\$decode_time
cal_\$decode_ascii_date:	decode an ASCII date .....	cal_\$decode_ascii_date
cal_\$decode_ascii_time:	decode an ascii time .....	cal_\$decode_ascii_time
cal_\$decode_ascii_tzdif:	decode an ASCII time zone .....	cal_\$decode_ascii_tzdif
counts ctm_\$release_pv:	decrement pixel value use .....	ctm_\$release_pv
file /create a	default magtape descriptor .....	mts_\$create_default_desc
pad_\$def_ptk:	define a program function key .....	pad_\$def_ptk
window pad_\$set_view:	define the pad region under a .....	pad_\$set_view
name_\$delete_directory:	delete a directory .....	name_\$delete_directory
name_\$delete_file:	delete a file .....	name_\$delete_file
pad_\$delete_frame: clear and	delete a frame .....	pad_\$delete_frame
name_\$drop_link:	delete a link .....	name_\$drop_link
pgm_\$del_arg:	delete a program argument .....	pgm_\$del_arg
ipc_\$delete:	delete an IPC socket .....	ipc_\$delete
ios_\$delete:	delete an object .....	ios_\$delete
mts_\$get_attr: get a magtape	descriptor attribute .....	mts_\$get_attr
mts_\$set_attr: set a magtape	descriptor attribute .....	mts_\$set_attr
/close a magtape	descriptor file .....	mts_\$close_desc
mts_\$copy_desc: copy a magtape	descriptor file .....	mts_\$copy_desc
/create a default magtape	descriptor file .....	mts_\$create_default_desc
mts_\$open_desc: open a magtape	descriptor file .....	mts_\$open_desc
/open a magtape	descriptor file .....	mts_\$open_desc_uid
mts_\$intro: using magtape	descriptor files .....	mts_\$intro
open on the same/ ios_\$equal:	determine if two streams are .....	ios_\$equal
task_\$tasking_enabled:	determine whether tasking is/ .....	task_\$tasking_enabled
tpad_\$set_mode: set pointing	device response/ .....	tpad_\$set_mode
ios_\$intro:	device-independent I/O .....	ios_\$intro
stream access to	directories ios_dir_\$intro: .....	ios_dir_\$intro
/extract a	directory entry .....	name_\$extract_data_lc
whether a stream is open on a	directory ios_dir_\$isa: test .....	ios_dir_\$isa

open an object in an open	directory ios_dir_Sopen: .....	ios_dir_Sopen
ios_dir_Sreaddir: read a	directory .....	ios_dir_Sreaddir
ios_dir_Srewinddir: rewind a	directory .....	ios_dir_Srewinddir
ios_dir_Sseekdir: seek in a	directory .....	ios_dir_Sseekdir
return a seek key for a	directory ios_dir_Stelldir: .....	ios_dir_Stelldir
the current working or naming	directory /get .....	ios_Sget_dir_lc
the current working or naming	directory /change .....	ios_Sset_dir
/create a	directory .....	name_Screate_directory
/delete a	directory .....	name_Sdelete_directory
get the naming	directory name_Sget_ndir_lc: .....	name_Sget_ndir_lc
get the working	directory name_Sget_wdir_lc: .....	name_Sget_wdir_lc
name_Sread_dir_lc: read a	directory .....	name_Sread_dir_lc
name_Sset_ndir: set the naming	directory .....	name_Sset_ndir
sets the working	directory name_Sset_wdir: .....	name_Sset_wdir
pm_Sget_home_txt: get the home	directory .....	pm_Sget_home_txt
pad_Sdm_cmd: execute a	Display Manager command .....	pad_Sdm_cmd
pad_Sintro: the	Display Manager .....	pad_Sintro
window /get the total	display region occupied by a .....	pad_Sinq_full_window
/find out the	display type .....	pad_Sinq_disp_type
a stream is open on a local	DM pad /test whether .....	pad_Sisa_dm_pad
vec_Sidot16: return the	dot product of two 16-bit/ .....	vec_Sidot16
integer/ vec_Sidot: return the	dot product of two 32-bit .....	vec_Sidot
vec_Sddot: return the	dot product of two/ .....	vec_Sddot
vec_Sdot: return the	dot product of two/ .....	vec_Sdot
vec_Sidot16_i: return the	dot product of two vectors in/ .....	vec_Sidot16_i
vec_Sidot_i: return the	dot product of two vectors in/ .....	vec_Sidot_i
vec_Sddot_i: return the	dot product of two vectors in/ .....	vec_Sddot_i
vec_Sdot_i: return the	dot product of two vectors in/ .....	vec_Sdot_i
scalar /multiply a vector in a	double-precision matrix by a .....	vec_Sdmult_constant_i
/copy a vector from a	double-precision matrix into a/ .....	vec_Sdp_sp_i
/copy a vector from one	double-precision matrix to/ .....	vec_Sdcopy_i
/add a scalar to a vector in a	double-precision matrix .....	vec_Sdadd_constant_i
/value in a vector from a	double-precision matrix .....	vec_Sdmax_i
the elements of a vector in a	double-precision matrix /sum .....	vec_Sdsum_i
/swap two vectors in a	double-precision matrix .....	vec_Sdswap_i
/zero a vector in a	double-precision matrix .....	vec_Sdzero_i
/add vectors in two	double-precision matrixes .....	vec_Sdadd_vector_i
dot product of two vectors in	double-precision matrixes /the .....	vec_Sddot_i
/multiply two 4x4	double-precision matrixes .....	vec_Sdmat_mult
vec_Sdmat_multn: multiply two	double-precision matrixes .....	vec_Sdmat_multn
vec_Sdpostmult: multiply a	double-precision vector by a/ .....	vec_Sdpostmult
vec_Sdpremult: multiply a	double-precision vector by a/ .....	vec_Sdpremult
vec_Sdpostmultn: multiply a	double-precision vector by a/ .....	vec_Sdpostmultn
vec_Sdpremultn: multiply a	double-precision vector by a/ .....	vec_Sdpremultn
scalar /multiply a	double-precision vector by a .....	vec_Sdmult_constant
/a single-precision vector to a	double-precision vector in/ .....	vec_Ssp_dp_i
vec_Sdp_sp: copy a	double-precision vector to a/ .....	vec_Sdp_sp
another /scale and add one	double-precision vector to .....	vec_Sdmult_add
/add a scalar to a	double-precision vector .....	vec_Sdadd_constant
vec_Sdcopy: copy a	double-precision vector .....	vec_Sdcopy
vec_Sdinit: initialize a	double-precision vector .....	vec_Sdinit

maximum absolute value in a	double-precision vector /the	vec_Sdmax
/sum the elements of a	double-precision vector	vec_Sdsum
vec_Sdzero: zero a	double-precision vector	vec_Sdzero
a single-precision vector to a	double-precision vector /copy	vec_Ssp_dp
matrixes /scale and add	double-precision vectors in	vec_Sdmult_add_i
vec_Sdsub_i: subtract	double-precision vectors in/	vec_Sdsub_i
vec_Sdadd_vector: add two	double-precision vectors	vec_Sdadd_vector
/return the dot product of two	double-precision vectors	vec_Sddot
vec_Sdsub: subtract	double-precision vectors	vec_Sdsub
vec_Sdswap: swap two	double-precision vectors	vec_Sdswap
ios_Sdup:	duplicate a stream	ios_Sdup
make a noise of a specified	duration tone_Stime:	tone_Stime
rws_Sintro:	dynamic storage allocation	rws_Sintro
eventcount	ec2_Sadvance: increment an	ec2_Sadvance
eventcount	ec2_Sinit: initialize an	ec2_Sinit
Manager	ec2_Sintro: Level 2 Eventcount	ec2_Sintro
eventcount value	ec2_Sread: return an	ec2_Sread
eventcount trigger	ec2_Swait: wait for an	ec2_Swait
eventcount trigger during I/O	ec2_Swait_slow_io: wait for an	ec2_Swait_slow_io
eventcount trigger or/	ec2_Swait_svc: wait for an	vec_Swait_svc
current site prf_Sedit_job:	edit a print job at the	prf_Sedit_job
pad_Sedit_wait: wait until an	edit window closes	pad_Sedit_wait
procl_Sget_cput: get	elapsed CPU time	procl_Sget_cput
vector vec_Sisum16: sum the	elements of a 16-bit integer	vec_Sisum16
vector vec_Sisum: sum the	elements of a 32-bit integer	vec_Sisum
vector vec_Sdsum: sum the	elements of a double-precision	vec_Sdsum
vector vec_Ssum: sum the	elements of a single-precision	vec_Ssum
16-bit/ vec_Sisum16_i: sum the	elements of a vector in a	vec_Sisum16_i
32-bit/ vec_Sisum_i: sum the	elements of a vector in a	vec_Sisum_i
vec_Sdsum_i: sum the	elements of a vector in a/	vec_Sdsum_i
vec_Ssum_i: sum the	elements of a vector in a/	vec_Ssum_i
pfm_Senable:	enable asynchronous faults	pfm_Senable
pad_Scpr_enable:	enable cursor position reports	pad_Scpr_enable
/determine whether tasking is	enabled	task_Stasking_enabled
cal_Sencode_time:	encode a date and time	cal_Sencode_time
ios_Sseek_to_eof: seek to the	end of an object	ios_Sseek_to_eof
loader_Scond_load:	ensure a module is loaded	loader_Scond_load
extract a directory	entry name_Sextract_data_lc:	name_Sextract_data_lc
error_Scode: return the	error code from a completion/	error_Scode
error_Sprint: print an	error message	error_Sprint
/print a standard	error message	error_Sstd_format
error_Sprint_format: print an	error message in standard/	error_Sprint_format
/initialize standard	error reporting	error_Sinit_std_format
error_Sintro: Domain/OS	error reporting	error_Sintro
error_Sfind_text: find	error reporting text	error_Sfind_text
error_Sget_text: get	error reporting text	error_Sget_text
error_Sprint_name: print	error text with a name	error_Sprint_name
code from a completion status	error_Scode: return the error	error_Scode
in a completion status	error_Sfail: test the fail bit	error_Sfail
reporting text	error_Sfind_text: find error	error_Sfind_text
reporting text	error_Sget_text: get error	error_Sget_text

initialize standard error/	error_\$init_std_format: .....	error_\$init_std_format
reporting	error_\$intro: Domain/OS error .....	error_\$intro
module code from a completion/	error_\$module: return the .....	error_\$module
message	error_\$print: print an error .....	error_\$print
error message in standard/	error_\$print_format: print an .....	error_\$print_format
text with a name	error_\$print_name: print error .....	error_\$print_name
standard error message	error_\$std_format: print a .....	error_\$std_format
subsystem code from a/	error_\$subsys: return the .....	error_\$subsys
pfm_\$cleanup:	establish a clean-up handler .....	pfm_\$cleanup
pfm_\$establish_fault_handler:	establish a fault handler .....	pfm_\$establish_fault_handler
pad_\$set_full_window:	establish a window region .....	pad_\$set_full_window
icon/ pad_\$set_icon_pos:	establish an icon position and .....	pad_\$set_icon_pos
ec2_\$advance: increment an	eventcount .....	ec2_\$advance
ec2_\$init: initialize an	eventcount .....	ec2_\$init
get a pointer to a stream	eventcount ios_\$get_ec: .....	ios_\$get_ec
ipc_\$get_ec: get an IPC	eventcount .....	ipc_\$get_ec
ec2_\$intro: Level 2	Eventcount Manager .....	ec2_\$intro
get a pointer to a mailbox	eventcount mbx_\$get_ec: .....	mbx_\$get_ec
pgm_\$get_ec: get a process	eventcount .....	pgm_\$get_ec
task_\$get_ec: get a completion	eventcount .....	task_\$get_ec
get a pointer to the time	eventcount time_\$get_ec: .....	time_\$get_ec
ec2_\$wait_slow_io: wait for an	eventcount trigger during I/O .....	ec2_\$wait_slow_io
ec2_\$wait: wait for an	eventcount trigger .....	ec2_\$wait
ec2_\$wait_svc: wait for an	eventcount trigger or/ .....	ec2_\$wait_svc
ec2_\$read: return an	eventcount value .....	ec2_\$read
mutex_\$lock: obtain a mutual	exclusion lock .....	mutex_\$lock
release a mutual	exclusion lock mutex_\$unlock: .....	mutex_\$unlock
/initialize a mutual	exclusion lock record .....	mutex_\$init
mutex_\$intro: mutually	exclusive resource locking .....	mutex_\$intro
command pad_\$dm_cmd:	execute a Display Manager .....	pad_\$dm_cmd
ios_\$open: open an	existing object .....	ios_\$open
pbufs_\$open: open an	existing paste buffer .....	pbufs_\$open
pgm_\$exit:	exit a program .....	pgm_\$exit
task_\$exit:	exit a task .....	task_\$exit
pgm_\$set_severity: set the	exit severity level .....	pgm_\$set_severity
/wait until an icon	expands or moves .....	pad_\$icon_wait
name_\$extract_data_lc:	extract a directory entry .....	name_\$extract_data_lc
pad_\$set_scale: set a scale	factor for cursor operations .....	pad_\$set_scale
status error_\$fail: test the	fail bit in a completion .....	error_\$fail
pfm_\$error_trap: simulate a	fault and save a traceback .....	pfm_\$error_trap
fault_\$intro: Domain/OS	fault codes .....	fault_\$intro
trigger or asynchronous	fault /wait for an eventcount .....	ec2_\$wait_svc
/establish a	fault handler .....	pfm_\$establish_fault_handler
/release a	fault handler .....	pfm_\$release_fault_handler
codes	fault_\$intro: Domain/OS fault .....	fault_\$intro
enable asynchronous	faults pfm_\$enable: .....	pfm_\$enable
inhibit asynchronous	faults pfm_\$inhibit: .....	pfm_\$inhibit
pfm_\$intro: managing	faults .....	pfm_\$intro
ios_\$inq_file_attr: get	file attributes .....	ios_\$inq_file_attr
ms_\$attributes: get the	file attributes of a mapped/ .....	ms_\$attributes
prf_\$name_print: spool a	file for printing .....	prf_\$name_print

close a magtape descriptor	file mts_\$close_desc: .....	mts_\$close_desc
copy a magtape descriptor	file mts_\$copy_desc: .....	mts_\$copy_desc
a default magtape descriptor	file /create .....	mts_\$create_default_desc
open a magtape descriptor	file mts_\$open_desc: .....	mts_\$open_desc
open a magtape descriptor	file mts_\$open_desc_uid: .....	mts_\$open_desc_uid
create a permanent	file name_\$create_file: .....	name_\$create_file
name_\$delete_file: delete a	file .....	name_\$delete_file
set print options from a	file prf_\$config_file: .....	prf_\$config_file
a mapped object with the	file system /synchronize .....	ms_\$fw_file
a mapped object with the	file system /synchronize .....	ms_\$fw_partial
ms_\$intro: mapping	file system objects .....	ms_\$intro
using magtape descriptor	files mts_\$intro: .....	mts_\$intro
ctm_\$find_color:	find a color in the color map .....	ctm_\$find_color
error_\$find_text:	find error reporting text .....	error_\$find_text
font pad_\$inq_font:	find out about the current .....	pad_\$inq_font
pad_\$inq_kbd:	find out about the keyboard .....	pad_\$inq_kbd
are on the/ ms_\$neighbors:	find out if two mapped objects .....	ms_\$neighbors
pad_\$inq_disp_type:	find out the display type .....	pad_\$inq_disp_type
the window pad_\$inq_view:	find out the pad region under .....	pad_\$inq_view
an icon pad_\$is_icon:	find out whether a window is .....	pad_\$is_icon
value in a/ vec_\$imax16:	find the maximum absolute .....	vec_\$imax16
value in a 32-bit/ vec_\$imax:	find the maximum absolute .....	vec_\$imax
value in a/ vec_\$dmax:	find the maximum absolute .....	vec_\$dmax
value in a/ vec_\$smax:	find the maximum absolute .....	vec_\$smax
value in a/ vec_\$imax16_i:	find the maximum absolute .....	vec_\$imax16_i
value in a/ vec_\$imax_i:	find the maximum absolute .....	vec_\$imax_i
value in a/ vec_\$dmax_i:	find the maximum absolute .....	vec_\$dmax_i
value in a vector/ vec_\$smax_i:	find the maximum absolute .....	vec_\$smax_i
pad_\$inq_position:	find the output cursor .....	pad_\$inq_position
fpp_\$control: modify the	floating-point control/ .....	fpp_\$control
fpp_\$intro: the	Floating-Point Package .....	fpp_\$intro
fpp_\$restore: load a new	floating-point state .....	fpp_\$restore
fpp_\$save: save the	floating-point state .....	fpp_\$save
the current and load a new	floating-point state /save .....	fpp_\$save_restore
/return the size of a	floating-point state .....	fpp_\$save_restore_size
fpp_\$status: modify the	floating-point status register .....	fpp_\$status
find out about the current	font pad_\$inq_font: .....	pad_\$inq_font
get the current icon	font pad_\$inq_icon_font: .....	pad_\$inq_icon_font
load a character	font pad_\$load_font: .....	pad_\$load_font
choose a new icon	font pad_\$set_icon_font: .....	pad_\$set_icon_font
use a new loaded	font pad_\$use_font: .....	pad_\$use_font
an error message in standard	format /print .....	error_\$print_format
vfmt_\$rs10:	formatted read from a stream .....	vfmt_\$rs10
vfmt_\$rs2:	formatted read from a stream .....	vfmt_\$rs2
vfmt_\$rs5:	formatted read from a stream .....	vfmt_\$rs5
vfmt_\$decode10:	formatted read from a string .....	vfmt_\$decode10
vfmt_\$decode2:	formatted read from a string .....	vfmt_\$decode2
vfmt_\$decode5:	formatted read from a string .....	vfmt_\$decode5
input vfmt_\$read10:	formatted read from standard .....	vfmt_\$read10
input vfmt_\$read2:	formatted read from standard .....	vfmt_\$read2
input vfmt_\$read5:	formatted read from standard .....	vfmt_\$read5

vfmt_\$ws:	formatted write to a stream	vfmt_\$ws
vfmt_\$ws10:	formatted write to a stream	vfmt_\$ws10
vfmt_\$ws2:	formatted write to a stream	vfmt_\$ws2
vfmt_\$ws5:	formatted write to a stream	vfmt_\$ws5
vfmt_\$encode:	formatted write to a string	vfmt_\$encode
vfmt_\$encode10:	formatted write to a string	vfmt_\$encode10
vfmt_\$encode2:	formatted write to a string	vfmt_\$encode2
vfmt_\$encode5:	formatted write to a string	vfmt_\$encode5
output vfmt_\$write:	formatted write to standard	vfmt_\$write
output vfmt_\$write10:	formatted write to standard	vfmt_\$write10
output vfmt_\$write2:	formatted write to standard	vfmt_\$write2
output vfmt_\$write5:	formatted write to standard	vfmt_\$write5
vfmt_\$intro:	variable formatting package	vfmt_\$intro
floating-point control/ Package	fpp_\$control: modify the Package fpp_\$intro: the Floating-Point	fpp_\$control fpp_\$intro
floating-point state	fpp_\$restore: load a new	fpp_\$restore
floating-point state	fpp_\$save: save the	fpp_\$save
current and load a new/ the size of a floating-point/ floating-point status/ pad_\$clear_frame: clear a	fpp_\$save_restore: save the fpp_\$save_restore_size: return fpp_\$status: modify the frame	fpp_\$save_restore fpp_\$save_restore_size fpp_\$status pad_\$clear_frame
pad_\$close_frame: close a	frame	pad_\$close_frame
pad_\$create_frame: create a	frame	pad_\$create_frame
clear and delete a	frame pad_\$delete_frame:	pad_\$delete_frame
/absolute value in a vector	from a 16-bit integer matrix	vec_\$imax16_i
/absolute value in a vector	from a 32-bit integer matrix	vec_\$imax_i
/return the error code	from a completion status	error_\$code
/return the module code	from a completion status	error_\$module
/return the subsystem code	from a completion status	error_\$subsys
vec_\$dp_sp_i: copy a vector	from a double-precision matrix/	vec_\$dp_sp_i
/absolute value in a vector	from a double-precision matrix	vec_\$dmax_i
set print options	from a file prf_\$config_file:	prf_\$config_file
/restore a bitmap	from a GMF	gmf_\$restore_plane
mbx_\$get_rec: get a message	from a mailbox	mbx_\$get_rec
/allocate read/write storage	from a pool	rws_\$alloc_rw_pool
/absolute value in a vector	from a single-precision matrix	vec_\$max_i
ios_\$get: read data	from a stream	ios_\$get
prf_\$stream_print: print	from a stream	prf_\$stream_print
vfmt_\$rs10: formatted read	from a stream	vfmt_\$rs10
vfmt_\$rs2: formatted read	from a stream	vfmt_\$rs2
vfmt_\$rs5: formatted read	from a stream	vfmt_\$rs5
vfmt_\$decode10: formatted read	from a string	vfmt_\$decode10
vfmt_\$decode2: formatted read	from a string	vfmt_\$decode2
vfmt_\$decode5: formatted read	from a string	vfmt_\$decode5
vec_\$copy16_i: copy a vector	from one 16-bit integer matrix/	vec_\$copy16_i
vec_\$copy_i: copy a vector	from one 32-bit integer matrix/	vec_\$copy_i
vec_\$dcopy_i: copy a vector	from one double-precision/	vec_\$dcopy_i
vec_\$scopy_i: copy a vector	from one single-precision/	vec_\$scopy_i
vfmt_\$read10: formatted read	from standard input	vfmt_\$read10
vfmt_\$read2: formatted read	from standard input	vfmt_\$read2
vfmt_\$read5: formatted read	from standard input	vfmt_\$read5

	seek with a	full key ios_\$seek_full_key:	ios_\$seek_full_key
	name_\$get_path_lc: get a	full pathname	name_\$get_path_lc
	ios_\$inq_full_key: return a	full seek key	ios_\$inq_full_key
	make a window visible and	full-size pad_\$select_window:	pad_\$select_window
pad_\$def_pfk:	define a program	function key	pad_\$def_pfk
	task_\$get_ec:	get a completion eventcount	task_\$get_ec
	name_\$get_path_lc:	get a full pathname	name_\$get_path_lc
the queue	prf_\$read_queue:	get a list of print jobs in	prf_\$read_queue
	prf_\$get_sites:	get a list of print sites	prf_\$get_sites
current/	prf_\$get_printers:	get a list of printers at the	prf_\$get_printers
	loader_\$inquire_stat:	get a load history summation	loader_\$inquire_stat
	attribute mts_\$get_attr:	get a magtape descriptor	mts_\$get_attr
	mbx_\$get_rec:	get a message from a mailbox	mbx_\$get_rec
eventcount	mbx_\$get_ec:	get a pointer to a mailbox	mbx_\$get_ec
eventcount	ios_\$get_ec:	get a pointer to a stream	ios_\$get_ec
	stream ios_\$locate:	get a pointer to data in a	ios_\$locate
eventcount	time_\$get_ec:	get a pointer to the time	time_\$get_ec
	prf_\$inq_option:	get a print request option	prf_\$inq_option
	pgm_\$get_ec:	get a process eventcount	pgm_\$get_ec
	pgm_\$get_puid:	get a process UID	pgm_\$get_puid
	pgm_\$get_arg:	get a program argument	pgm_\$get_arg
	ipc_\$get_ec:	get an IPC eventcount	ipc_\$get_ec
	ipc_\$resolve:	get an IPC socket handle	ipc_\$resolve
ios_\$inq_path_name_lc:	get an object's pathname	ios_\$inq_path_name_lc	ios_\$inq_path_name_lc
	proc1_\$get_cput:	get elapsed CPU time	proc1_\$get_cput
	error_\$get_text:	get error reporting text	error_\$get_text
	ios_\$inq_file_attr:	get file attributes	ios_\$inq_file_attr
	pad_\$inq_icon:	get icon information	pad_\$inq_icon
	loader_\$inquire_image:	get image information	loader_\$inquire_image
	task_\$get_info:	get information about a task	task_\$get_info
	current/ tpad_\$inquire:	get information about the	tpad_\$inquire
	loader_\$inquire_section:	get information on an image/	loader_\$inquire_section
	loader_\$inquire_known_lib:	get information on known/	loader_\$inquire_known_lib
information	proc2_\$get_info:	get level 2 process	proc2_\$get_info
	osinfo_\$get_rev:	get Domain/OS revision information	osinfo_\$get_rev
	sio_\$inquire:	get serial line options	sio_\$inquire
	pad_\$inq_icon_font:	get the current icon font	pad_\$inq_icon_font
naming/	ios_\$get_dir_lc:	get the current working or	ios_\$get_dir_lc
mapped object	ms_\$attributes:	get the file attributes of a	ms_\$attributes
	pm_\$get_home_txt:	get the home directory	pm_\$get_home_txt
	cal_\$decode_local_time:	get the local date and time	cal_\$decode_local_time
	cal_\$get_local_time:	get the local time	cal_\$get_local_time
	cal_\$get_info:	get the local time zone	cal_\$get_info
	name_\$get_ndir_lc:	get the naming directory	name_\$get_ndir_lc
	vector pgm_\$get_args:	get the program argument	pgm_\$get_args
	pm_\$get_sid_txt:	get the SID	pm_\$get_sid_txt
of/	loader_\$lookup_stack_size:	get the stack size requirement	loader_\$lookup_stack_size
	loader_\$lookup_start_addr:	get the start address of an/	loader_\$lookup_start_addr
	time_\$clock:	get the system clock value	time_\$clock
	pad_\$inq_full_window:	get the total display region/	pad_\$inq_full_window
	ios_\$inq_type_uid:	get the type of an object	ios_\$inq_type_uid

process	proc2_\$who_am_i:	get the UID of the calling	proc2_\$who_am_i
	name_\$get_wdir_lc:	get the working directory	name_\$get_wdir_lc
	/add to the Known	Global Table (KGT)	loader_\$kg_define
	gmf_\$close:	close a GMF	gmf_\$close
	copy a bitmap plane to a	GMF gmf_\$copy_plane:	gmf_\$copy_plane
	part of a bitmap plane to a	GMF gmf_\$copy_subplane: copy	gmf_\$copy_subplane
	gmf_\$open:	open a GMF	gmf_\$open
	restore a bitmap from a	GMF gmf_\$restore_plane:	gmf_\$restore_plane
		gmf_\$close: close a GMF	gmf_\$close
	plane to a GMF	gmf_\$copy_plane: copy a bitmap	gmf_\$copy_plane
	of a bitmap plane to a GMF	gmf_\$copy_subplane: copy part	gmf_\$copy_subplane
	metafiles	gmf_\$intro: using graphics	gmf_\$intro
		gmf_\$open: open a GMF	gmf_\$open
	bitmap from a GMF	gmf_\$restore_plane: restore a	gmf_\$restore_plane
	gmf_\$intro:	using graphics metafiles	gmf_\$intro
	convert a stream ID to a	handle ios_\$get_handle:	ios_\$get_handle
	get an IPC socket	handle ipc_\$resolve:	ipc_\$resolve
	return the task	handle task_\$get_handle:	task_\$get_handle
	establish a clean-up	handler pfm_\$cleanup:	pfm_\$cleanup
	/establish a fault	handler	pfm_\$establish_fault_handler
	/release a fault	handler	pfm_\$release_fault_handler
	reset a clean-up	handler pfm_\$reset_cleanup:	pfm_\$reset_cleanup
	release clean-up	handlers pfm_\$rls_cleanup:	pfm_\$rls_cleanup
rws_\$alloc_heap_pool:	allocate	heap storage	rws_\$alloc_heap_pool
	/release	heap storage	rws_\$release_heap_pool
	start a mailbox	helper mbx_\$start_helper:	mbx_\$start_helper
	pm_\$get_home_txt:	get the home directory	pm_\$get_home_txt
	/establish an icon position and	icon character	pad_\$set_icon_pos
pad_\$icon_wait:	wait until an	icon expands or moves	pad_\$icon_wait
	get the current	icon font pad_\$inq_icon_font:	pad_\$inq_icon_font
	choose a new	icon font pad_\$set_icon_font:	pad_\$set_icon_font
	pad_\$inq_icon:	get icon information	pad_\$inq_icon
new pad and associated window	icon /create a	icon	pad_\$create_icon
out whether a window is an	icon pad_\$is_icon: find	icon	pad_\$is_icon
change a window into an	icon pad_\$make_icon:	icon	pad_\$make_icon
character /establish an	icon position and icon	icon	pad_\$set_icon_pos
switch a stream to another	ID ios_\$switch:	ID	ios_\$switch
/convert a stream	ID to a handle	ID	ios_\$get_handle
loader_\$inquire_image:	get image information	image	loader_\$inquire_image
stack size requirement of an	image /get the	image	loader_\$lookup_stack_size
/get the start address of an	image	image	loader_\$lookup_start_addr
/get information on an	image section	image	loader_\$inquire_section
pad_\$force_prompt:	force an incomplete record to an input/	input	pad_\$force_prompt
ec2_\$advance:	increment an eventcount	eventcount	ec2_\$advance
counts ctm_\$inc_use_count:	increment pixel value use	pixel value use	ctm_\$inc_use_count
pfm_\$inhibit:	inhibit asynchronous faults	faults	pfm_\$inhibit
vector vec_\$iinit16:	initialize a 16-bit integer	integer	vec_\$iinit16
vector vec_\$iinit:	initialize a 32-bit integer	integer	vec_\$iinit
vector vec_\$dinit:	initialize a double-precision	precision	vec_\$dinit
lock record mutex_\$init:	initialize a mutual exclusion	mutex	mutex_\$init
vector vec_\$sinit:	initialize a single-precision	precision	vec_\$sinit

ec2_\$init:	initialize an eventcount .....	ec2_\$init
options prf_\$init:	initialize print request .....	prf_\$init
error_\$init_std_format:	initialize standard error/ .....	error_\$init_std_format
an incomplete record to an	input pad /force .....	pad_\$force_prompt
formatted read from standard	input vfmt_\$read10: .....	vfmt_\$read10
formatted read from standard	input vfmt_\$read2: .....	vfmt_\$read2
formatted read from standard	input vfmt_\$read5: .....	vfmt_\$read5
loader_\$clear_inlib: clear the	installed library list .....	loader_\$clear_inlib
/create a privately	installed library .....	loader_\$inlib
/multiply a vector in a 16-bit	integer matrix by a scalar .....	vec_\$imult_constant16_i
/multiply a vector in a 32-bit	integer matrix by a scalar .....	vec_\$imult_constant_i
/copy a vector from one 16-bit	integer matrix to another .....	vec_\$icopy16_i
/copy a vector from one 32-bit	integer matrix to another .....	vec_\$icopy_i
scalar to a vector in a 16-bit	integer matrix /add a .....	vec_\$iadd_constant16_i
scalar to a vector in a 32-bit	integer matrix /add a .....	vec_\$iadd_constant_i
in a vector from a 16-bit	integer matrix /absolute value .....	vec_\$imax16_i
in a vector from a 32-bit	integer matrix /absolute value .....	vec_\$imax_i
of a vector in a 16-bit	integer matrix /the elements .....	vec_\$isum16_i
of a vector in a 32-bit	integer matrix /the elements .....	vec_\$isum_i
/swap two vectors in a 16-bit	integer matrix .....	vec_\$iswap16_i
swap two vectors in a 32-bit	integer matrix vec_\$iswap_i: .....	vec_\$iswap_i
/zero a vector in a 16-bit	integer matrix .....	vec_\$izero16_i
zero a vector in a 32-bit	integer matrix vec_\$izero_i: .....	vec_\$izero_i
/add vectors in two 16-bit	integer matrixes .....	vec_\$iadd_vector16_i
/add vectors in two 32-bit	integer matrixes .....	vec_\$iadd_vector_i
of two vectors in 16-bit	integer matrixes /dot product .....	vec_\$idot16_i
of two vectors in 32-bit	integer matrixes /dot product .....	vec_\$idot_i
/multiply two 4x4 32-bit	integer matrixes .....	vec_\$imat_mult
/multiply two 4x4 16-bit	integer matrixes .....	vec_\$imat_mult16
/multiply two 32-bit	integer matrixes .....	vec_\$imat_multn
/multiply two 16-bit	integer matrixes .....	vec_\$imat_multn16
matrix /multiply a 32-bit	integer vector by a 4x4 .....	vec_\$ipostmult
matrix /multiply a 16-bit	integer vector by a 4x4 .....	vec_\$ipostmult16
/multiply a 32-bit	integer vector by a matrix .....	vec_\$ipostmultn
/multiply a 16-bit	integer vector by a matrix .....	vec_\$ipostmultn16
/multiply a 32-bit	integer vector by a matrix .....	vec_\$ipremultn
/multiply a 16-bit	integer vector by a matrix .....	vec_\$ipremultn16
/multiply a 32-bit	integer vector by a scalar .....	vec_\$imult_constant
/multiply a 16-bit	integer vector by a scalar .....	vec_\$imult_constant16
/scale and add one 32-bit	integer vector to another .....	vec_\$imult_add
/scale and add one 16-bit	integer vector to another .....	vec_\$imult_add16
a scalar constant to a 32-bit	integer vector /add .....	vec_\$iadd_constant
a scalar constant to a 16-bit	integer vector /add .....	vec_\$iadd_constant16
vec_\$icopy: copy a 32-bit	integer vector .....	vec_\$icopy
vec_\$icopy16: copy a 16-bit	integer vector .....	vec_\$icopy16
initialize a 32-bit	integer vector vec_\$iinit: .....	vec_\$iinit
initialize a 16-bit	integer vector vec_\$iinit16: .....	vec_\$iinit16
absolute value in a 32-bit	integer vector /the maximum .....	vec_\$imax
absolute value in a 16-bit	integer vector /the maximum .....	vec_\$imax16
sum the elements of a 32-bit	integer vector vec_\$isum: .....	vec_\$isum
sum the elements of a 16-bit	integer vector vec_\$isum16: .....	vec_\$isum16

vec_\$zero: zero a 32-bit	integer vector .....	vec_\$zero
vec_\$zero16: zero a 16-bit	integer vector .....	vec_\$zero16
vec_\$sub16_i: subtract 16-bit	integer vectors in matrixes .....	vec_\$sub16_i
vec_\$sub_i: subtract 32-bit	integer vectors in matrixes .....	vec_\$sub_i
/add two 32-bit	integer vectors .....	vec_\$iadd_vector
/add two 16-bit	integer vectors .....	vec_\$iadd_vector16
the dot product of two 32-bit	integer vectors /return .....	vec_\$idot
the dot product of two 16-bit	integer vectors /return .....	vec_\$idot16
vec_\$sub: subtract 32-bit	integer vectors .....	vec_\$sub
vec_\$sub16: subtract 16-bit	integer vectors .....	vec_\$sub16
vec_\$swap: swap two 32-bit	integer vectors .....	vec_\$swap
vec_\$swap16: swap two 16-bit	integer vectors .....	vec_\$swap16
ipc_\$intro: Domain/OS	Interprocess Communication .....	ipc_\$intro
time_\$wait: wait for an	interval .....	time_\$wait
/make a window	invisible .....	pad_\$make_invisible
pgm_\$invoke: invoke a program	.....	pgm_\$invoke
an eventcount trigger during	I/O /wait for .....	ec2_\$wait_slow_io
ios_\$intro: device-independent	I/O .....	ios_\$intro
sio_\$intro: controlling serial	I/O lines .....	sio_\$intro
an object's pathname	ios_\$change_path_name: change .....	ios_\$change_path_name
	ios_\$close: close a stream .....	ios_\$close
and open a stream to it	ios_\$create: create an object .....	ios_\$create
	ios_\$delete: delete an object .....	ios_\$delete
to directories	ios_dir_\$intro: stream access .....	ios_dir_\$intro
stream is open on a directory	ios_dir_\$isa: test whether a .....	ios_dir_\$isa
in an open directory	ios_dir_\$open: open an object .....	ios_dir_\$open
directory	ios_dir_\$readdir: read a .....	ios_dir_\$readdir
directory	ios_dir_\$rewinddir: rewind a .....	ios_dir_\$rewinddir
directory	ios_dir_\$seekdir: seek in a .....	ios_dir_\$seekdir
seek key for a directory	ios_dir_\$stelldir: return a .....	ios_dir_\$stelldir
	ios_\$dup: duplicate a stream .....	ios_\$dup
streams are open on the same/	ios_\$equal: determine if two .....	ios_\$equal
object to permanent storage	ios_\$force_write_file: save an .....	ios_\$force_write_file
stream	ios_\$get: read data from a .....	ios_\$get
current working or naming/	ios_\$get_dir_lc: get the .....	ios_\$get_dir_lc
a stream eventcount	ios_\$get_ec: get a pointer to .....	ios_\$get_ec
stream ID to a handle	ios_\$get_handle: convert a .....	ios_\$get_handle
byte position of a stream/	ios_\$inq_byte_pos: return the .....	ios_\$inq_byte_pos
connection attributes	ios_\$inq_conn_flags: return .....	ios_\$inq_conn_flags
the length of the next record	ios_\$inq_cur_rec_len: return .....	ios_\$inq_cur_rec_len
attributes	ios_\$inq_file_attr: get file .....	ios_\$inq_file_attr
full seek key	ios_\$inq_full_key: return a .....	ios_\$inq_full_key
manager attributes	ios_\$inq_mgr_flags: return .....	ios_\$inq_mgr_flags
object attributes	ios_\$inq_obj_flags: return .....	ios_\$inq_obj_flags
object's pathname	ios_\$inq_path_name_lc: get an .....	ios_\$inq_path_name_lc
record position of a stream/	ios_\$inq_rec_pos: return the .....	ios_\$inq_rec_pos
the number of bytes left in a/	ios_\$inq_rec_remainder: return .....	ios_\$inq_rec_remainder
record type of an object	ios_\$inq_rec_type: return the .....	ios_\$inq_rec_type
short seek key	ios_\$inq_short_key: return a .....	ios_\$inq_short_key
type of an object	ios_\$inq_type_uid: get the .....	ios_\$inq_type_uid
I/O	ios_\$intro: device-independent .....	ios_\$intro

data in a stream	ios_\$locate: get a pointer to .....	ios_\$locate
object	ios_\$open: open an existing .....	ios_\$open
	ios_\$put: write to an object .....	ios_\$put
stream	ios_\$replicate: replicate a .....	ios_\$replicate
	ios_\$seek: seek on a stream .....	ios_\$seek
a full key	ios_\$seek_full_key: seek with .....	ios_\$seek_full_key
a short key	ios_\$seek_short_key: seek with .....	ios_\$seek_short_key
beginning of an object	ios_\$seek_to_bof: seek to the .....	ios_\$seek_to_bof
end of an object	ios_\$seek_to_eof: seek to the .....	ios_\$seek_to_eof
connection attributes	ios_\$set_conn_flag: set .....	ios_\$set_conn_flag
current working or naming/ set the locate buffer size	ios_\$set_dir: change the .....	ios_\$set_dir
attributes	ios_\$set_locate_buffer_size: .....	ios_\$set_locate_buffer_size
object's record type	ios_\$set_obj_flag: set object .....	ios_\$set_obj_flag
to another ID	ios_\$set_rec_type: change an .....	ios_\$set_rec_type
object at the current stream/ ipc_\$sar: send an	ios_\$switch: switch a stream .....	ios_\$switch
ipc_\$rcv: retrieve an	ios_\$truncate: truncate an .....	ios_\$truncate
ipc_\$send: send an	IPC datagram and await a reply .....	ipc_\$sar
ipc_\$wait: wait for an	IPC datagram .....	ipc_\$rcv
ipc_\$get_ec: get an	IPC datagram .....	ipc_\$send
ipc_\$resolve: get an	IPC eventcount .....	ipc_\$wait
ipc_\$close: close an	IPC socket handle .....	ipc_\$get_ec
ipc_\$create: create an	IPC socket .....	ipc_\$resolve
ipc_\$delete: delete an	IPC socket .....	ipc_\$close
ipc_\$open: open an	IPC socket .....	ipc_\$create
socket	IPC socket .....	ipc_\$delete
socket	ipc_\$close: close an IPC .....	ipc_\$open
socket	ipc_\$create: create an IPC .....	ipc_\$close
eventcount	ipc_\$delete: delete an IPC .....	ipc_\$create
Communication	ipc_\$get_ec: get an IPC .....	ipc_\$delete
	ipc_\$intro: Domain/OS Interprocess ....	ipc_\$get_ec
	ipc_\$open: open an IPC socket .....	ipc_\$intro
	datagram	ipc_\$open
	socket handle	ipc_\$rcv: retrieve an IPC .....
and await a reply	ipc_\$resolve: get an IPC .....	ipc_\$rcv
datagram	ipc_\$sar: send an IPC datagram .....	ipc_\$resolve
datagram	ipc_\$send: send an IPC .....	ipc_\$sar
/return a seek	ipc_\$wait: wait for an IPC .....	ipc_\$send
return a full seek	key for a directory .....	ipc_\$wait
return a short seek	key ios_\$inq_full_key: .....	ios_dir_\$stlldir
seek with a full	key ios_\$inq_short_key: .....	ios_\$inq_full_key
seek with a short	key ios_\$seek_full_key: .....	ios_\$inq_short_key
define a program function	key ios_\$seek_short_key: .....	ios_\$seek_full_key
pad_\$locate: locate the	key pad_\$def_pfk: .....	ios_\$seek_short_key
find out about the	keyboard cursor .....	pad_\$def_pfk
add to the Known Global Table	keyboard pad_\$inq_kbd: .....	pad_\$locate
task_\$blast:	(KGT) loader_\$kg_define: .....	pad_\$inq_kbd
mts_\$label:	kill a task without cleanup .....	loader_\$kg_define
name_\$cname: change a	label a magtape .....	task_\$blast
/return the	leaf name .....	mts_\$label
proc1_\$intro: the	length of the next record .....	name_\$cname
	Level 1 Process Manager .....	ios_\$inq_cur_rec_len
		proc1_\$intro

ec2_Sintro:	Level 2 Eventcount Manager .....	ec2_Sintro
proc2_\$get_info: get	level 2 process information .....	proc2_\$get_info
proc2_Sintro: the	Level 2 Process Manager .....	proc2_Sintro
proc2_Slist: list	level 2 process UIDs .....	proc2_Slist
protected subsystem privilege	level aclm_Sintro: change the .....	aclm_Sintro
set the exit severity	level pgm_\$set_severity: .....	pgm_\$set_severity
manipulate predefined large/	lib_Sintro: calls to .....	lib_Sintro
/get information on known	libraries .....	loader_Sinquire_known_lib
/clear the installed	library list .....	loader_Sclear_inlib
create a privately installed	library loader_Sinlib: .....	loader_Sinlib
prf_Sintro: the Domain/OS print	library .....	prf_Sintro
task_Sintro: the Domain/OS task	library .....	task_Sintro
vec_Sintro: the Vector	Library .....	vec_Sintro
sio_Scontrol: set serial	line options .....	sio_Scontrol
sio_Sinquire: get serial	line options .....	sio_Sinquire
controlling serial I/O	lines sio_Sintro: .....	sio_Sintro
name_\$add_link: create a	link .....	name_\$add_link
name_\$drop_link: delete a	link .....	name_\$drop_link
the pathname associated with a	link /return .....	name_\$read_link_lc
proc2_Slist:	list level 2 process UIDs .....	proc2_Slist
clear the installed library	list loader_Sclear_inlib: .....	loader_Sclear_inlib
queue prf_Sread_queue: get a	list of print jobs in the .....	prf_Sread_queue
prf_\$get_sites: get a	list of print sites .....	prf_\$get_sites
prf_\$get_printers: get a	list of printers at the/ .....	prf_\$get_printers
pad_Sinq_windows:	list the windows on a pad .....	pad_Sinq_windows
pad_Sload_font:	load a character font .....	pad_Sload_font
loader_Sload:	load a module .....	loader_Sload
state fpp_Srestore:	load a new floating-point .....	fpp_Srestore
state /save the current and	load a new floating-point .....	fpp_Ssave_restore
loader_Sinquire_stat: get a	load history summation .....	loader_Sinquire_stat
pad_Suse_font: use a new	loaded font .....	pad_Suse_font
ensure a module is	loaded loader_Scond_load: .....	loader_Scond_load
loader_Sintro: the Domain/OS	loader .....	loader_Sintro
installed library list	loader_Sclear_inlib: clear the .....	loader_Sclear_inlib
module is loaded	loader_Scond_load: ensure a .....	loader_Scond_load
privately installed library	loader_Sinlib: create a .....	loader_Sinlib
image information	loader_Sinquire_image: get .....	loader_Sinquire_image
information on known/	loader_Sinquire_known_lib: get .....	loader_Sinquire_known_lib
information on an image/	loader_Sinquire_section: get .....	loader_Sinquire_section
load history summation	loader_Sinquire_stat: get a .....	loader_Sinquire_stat
loader	loader_Sintro: the Domain/OS .....	loader_Sintro
Known Global Table (KGT)	loader_Skg_define: add to the .....	loader_Skg_define
symbol	loader_Skg_lookup: look up a .....	loader_Skg_lookup
loader_Sload: load a module	.....	loader_Sload
the stack size requirement of/	loader_Slookup_stack_size: get .....	loader_Slookup_stack_size
the start address of an image	loader_Slookup_start_addr: get .....	loader_Slookup_start_addr
/set the	locate buffer size .....	ios_Sset_locate_buffer_size
pad_Slocate:	locate the keyboard cursor .....	pad_Slocate
mode /re-establish the	locator origin in relative .....	tpad_Sset_cursor
information about the current	locator response /get .....	tpad_Sinquire
calls tpad_Sintro:	locator (touchpad) manager .....	tpad_Sintro

	return the last locator used	tpad_\$inq_dtype: .....	tpad_\$inq_dtype
ms_\$scrtmp:	create, map, and lock a temporary object	.....	ms_\$scrtmp
ms_\$scmapl:	create, map, and lock an object	.....	ms_\$scmapl
ms_\$smapl:	map and lock an object	.....	ms_\$smapl
	obtain a mutual exclusion lock	mutex_\$lock: .....	mutex_\$lock
	release a mutual exclusion lock	mutex_\$unlock: .....	mutex_\$unlock
ms_\$relock:	change the lock on an object	.....	ms_\$relock
	initialize a mutual exclusion lock record	mutex_\$init: .....	mutex_\$init
	mutually exclusive resource locking	mutex_\$intro: .....	mutex_\$intro
mts_\$get_attr:	get a magtape descriptor attribute	.....	mts_\$get_attr
mts_\$set_attr:	set a magtape descriptor attribute	.....	mts_\$set_attr
mts_\$close_desc:	close a magtape descriptor file	.....	mts_\$close_desc
mts_\$copy_desc:	copy a magtape descriptor file	.....	mts_\$copy_desc
	/create a default magtape descriptor file	.....	mts_\$create_default_desc
mts_\$open_desc:	open a magtape descriptor file	.....	mts_\$open_desc
mts_\$open_desc_uid:	open a magtape descriptor file	.....	mts_\$open_desc_uid
	mts_\$intro: using magtape descriptor files	.....	mts_\$intro
	mts_\$label: label a magtape	.....	mts_\$label
mbx_\$get_rec_chan:	read a mailbox channel	.....	mbx_\$get_rec_chan
	/get a pointer to a mailbox eventcount	.....	mbx_\$get_ec
mbx_\$start_helper:	start a mailbox helper	.....	mbx_\$start_helper
mbx_\$close:	close a channel or mailbox	.....	mbx_\$close
mbx_\$create_server:	create a mailbox	.....	mbx_\$create_server
	get a message from a mailbox	mbx_\$get_rec: .....	mbx_\$get_rec
	open a client channel in a mailbox	mbx_\$open: .....	mbx_\$open
mbx_\$intro:	using mailboxes	.....	mbx_\$intro
	duration	tone_\$time: .....	tone_\$time
temporary	tone_\$intro: make a noise	.....	tone_\$intro
ms_\$mk_temporary:	make a permanent object	.....	ms_\$mk_temporary
permanent	ms_\$mk_permanent: make a temporary object	.....	ms_\$mk_permanent
pad_\$make_invisible:	make a window invisible	.....	pad_\$make_invisible
full-size	pad_\$select_window: make a window visible and	.....	pad_\$select_window
ios_\$inq_mgr_flags:	return manager attributes	.....	ios_\$inq_mgr_flags
	locator (touchpad) manager calls	tpad_\$intro: .....	tpad_\$intro
pad_\$dm_cmd:	execute a Display Manager command	.....	pad_\$dm_cmd
ctm_\$intro:	Color Table Manager	.....	ctm_\$intro
ec2_\$intro:	Level 2 Eventcount Manager	.....	ec2_\$intro
pad_\$intro:	the Display Manager	.....	pad_\$intro
pgm_\$intro:	the program manager	.....	pgm_\$intro
pm_\$intro:	the Domain/OS Process Manager	.....	pm_\$intro
	the Level 1 Process Manager	proc1_\$intro: .....	proc1_\$intro
	the Level 2 Process Manager	proc2_\$intro: .....	proc2_\$intro
aclm_\$down:	deassert subsystem manager rights	.....	aclm_\$down
aclm_\$up:	assert subsystem manager rights	.....	aclm_\$up
pfm_\$intro:	managing faults	.....	pfm_\$intro
sets	lib_\$intro: calls to manipulate predefined large	.....	lib_\$intro
mapped object	ms_\$remap: map a different portion of a	.....	ms_\$remap
ms_\$mapl_stream:	map an object via its XOID	.....	ms_\$mapl_stream
object	ms_\$scrtmp: create, map, and lock a temporary	.....	ms_\$scrtmp
ms_\$scmapl:	create, map, and lock an object	.....	ms_\$scmapl
ms_\$smapl:	map and lock an object	.....	ms_\$smapl

find a color in the color	map	ctm_\$find_color: .....	ctm_\$find_color
ms_\$addmap:	map more of a object	ms_\$addmap .....	ms_\$addmap
/advise the system on	mapped object access	ms_\$advic .....	ms_\$advic
get the file attributes of a	mapped object	ms_\$attribut .....	ms_\$attribut
map a different portion of a	mapped object	ms_\$remap .....	ms_\$remap
ms_\$truncate: truncate a	mapped object	ms_\$truncate .....	ms_\$truncate
ms_\$fw_file: synchronize a	mapped object with the file/	ms_\$fw_file .....	ms_\$fw_file
/partially synchronize a	mapped object with the file/	ms_\$fw_parti .....	ms_\$fw_parti
ms_\$neighbors: find out if two	mapped objects are on the same/	ms_\$neighbor .....	ms_\$neighbor
ms_\$intro:	mapping file system objects	ms_\$intro .....	ms_\$intro
the byte position of a stream	marker /return	ios_\$inq_byte_pos .....	ios_\$inq_byte_pos
record position of a stream	marker /return the	ios_\$inq_rec_pos .....	ios_\$inq_rec_pos
object at the current stream	marker /truncate an	ios_\$truncate .....	ios_\$truncate
a vector in a double-precision	matrix by a scalar /multiply	vec_\$dmult_constant_i .....	vec_\$dmult_constant_i
a vector in a 16-bit integer	matrix by a scalar /multiply	vec_\$simult_constant16_i .....	vec_\$simult_constant16_i
a vector in a 32-bit integer	matrix by a scalar /multiply	vec_\$simult_constant_i .....	vec_\$simult_constant_i
a vector in a single-precision	matrix by a scalar /multiply	vec_\$smult_constant_i .....	vec_\$smult_constant_i
/vector from a double-precision	matrix into a single-precision/	vec_\$dp_sp_i .....	vec_\$dp_sp_i
from one single-precision	matrix to another /a vector	vec_\$copy_i .....	vec_\$copy_i
from one double-precision	matrix to another /a vector	vec_\$dcopy_i .....	vec_\$dcopy_i
vector from one 16-bit integer	matrix to another /copy a	vec_\$copy16_i .....	vec_\$copy16_i
vector from one 32-bit integer	matrix to another /copy a	vec_\$icopy_i .....	vec_\$icopy_i
a vector in a single-precision	matrix /add a scalar to	vec_\$add_constant_i .....	vec_\$add_constant_i
a vector in a double-precision	matrix /add a scalar to	vec_\$dadd_constant_i .....	vec_\$dadd_constant_i
vector from a double-precision	matrix /absolute value in a	vec_\$dmax_i .....	vec_\$dmax_i
vector by a 4x4	matrix /a double-precision	vec_\$dpostmult .....	vec_\$dpostmult
a double-precision vector by a	matrix /multiply	vec_\$dpostmultn .....	vec_\$dpostmultn
vector by a 4x4	matrix /a double-precision	vec_\$dpremult .....	vec_\$dpremult
a double-precision vector by a	matrix /multiply	vec_\$dpremultn .....	vec_\$dpremultn
matrix into a single-precision	matrix /a double-precision	vec_\$dp_sp_i .....	vec_\$dp_sp_i
a vector in a double-precision	matrix /sum the elements of	vec_\$dsum_i .....	vec_\$dsum_i
vectors in a double-precision	matrix	vec_\$dswap_i: swap two .....	vec_\$dswap_i
a vector in a double-precision	matrix	vec_\$dzero_i: zero .....	vec_\$dzero_i
a vector in a 16-bit integer	matrix /add a scalar to	vec_\$iadd_constant16_i .....	vec_\$iadd_constant16_i
a vector in a 32-bit integer	matrix /add a scalar to	vec_\$iadd_constant_i .....	vec_\$iadd_constant_i
a vector from a 16-bit integer	matrix /absolute value in	vec_\$imax16_i .....	vec_\$imax16_i
a vector from a 32-bit integer	matrix /absolute value in	vec_\$imax_i .....	vec_\$imax_i
integer vector by a 4x4	matrix /multiply a 32-bit	vec_\$ipostmult .....	vec_\$ipostmult
integer vector by a 4x4	matrix /multiply a 16-bit	vec_\$ipostmult16 .....	vec_\$ipostmult16
a 32-bit integer vector by a	matrix /multiply	vec_\$ipostmultn .....	vec_\$ipostmultn
a 16-bit integer vector by a	matrix /multiply	vec_\$ipostmultn16 .....	vec_\$ipostmultn16
a 32-bit vector by a 4x4	matrix /multiply	vec_\$ipremult .....	vec_\$ipremult
a 16-bit vector by a 4x4	matrix /multiply	vec_\$ipremult16 .....	vec_\$ipremult16
a 32-bit integer vector by a	matrix /multiply	vec_\$ipremultn .....	vec_\$ipremultn
a 16-bit integer vector by a	matrix /multiply	vec_\$ipremultn16 .....	vec_\$ipremultn16
a vector in a 16-bit integer	matrix /sum the elements of	vec_\$isum16_i .....	vec_\$isum16_i
a vector in a 32-bit integer	matrix /sum the elements of	vec_\$isum_i .....	vec_\$isum_i
vectors in a 16-bit integer	matrix /swap two	vec_\$iswap16_i .....	vec_\$iswap16_i
vectors in a 32-bit integer	matrix	vec_\$iswap_i: swap two .....	vec_\$iswap_i
a vector in a 16-bit integer	matrix	vec_\$iszero16_i: zero .....	vec_\$iszero16_i
a vector in a 32-bit integer	matrix	vec_\$iszero_i: zero .....	vec_\$iszero_i

vector from a single-precision	matrix /absolute value in a .....	vec_\$max_i
vector by a 4x4	matrix /a single-precision .....	vec_\$postmult
a single-precision vector by a	matrix /multiply .....	vec_\$postmultn
vector by a 4x4	matrix /a single-precision .....	vec_\$premult
a single-precision vector by a	matrix /multiply .....	vec_\$premultn
a vector in a single-precision	matrix /sum the elements of .....	vec_\$sum_i
vectors in a single-precision	matrix vec_\$swap_1: swap two .....	vec_\$swap_i
a vector in a single-precision	matrix vec_\$zero_i: zero .....	vec_\$zero_i
in two single-precision	matrixes /add vectors .....	vec_\$add_vector_i
in two double-precision	matrixes /add vectors .....	vec_\$dadd_vector_i
vectors in double-precision	matrixes /dot product of two .....	vec_\$dot_i
two 4x4 double-precision	matrixes /multiply .....	vec_\$dmat_mult
multiply two double-precision	matrixes vec_\$dmat_multn: .....	vec_\$dmat_multn
double-precision vectors in	matrixes /scale and add .....	vec_\$dmult_add_i
vectors in single-precision	matrixes /dot product of two .....	vec_\$dot_i
double-precision vectors in	matrixes /subtract .....	vec_\$dsub_i
vectors in two 16-bit integer	matrixes /add .....	vec_\$iadd_vector16_i
vectors in two 32-bit integer	matrixes /add .....	vec_\$iadd_vector_i
two vectors in 16-bit integer	matrixes /the dot product of .....	vec_\$idot16_i
two vectors in 32-bit integer	matrixes /the dot product of .....	vec_\$idot_i
two 4x4 32-bit integer	matrixes /multiply .....	vec_\$imat_mult
two 4x4 16-bit integer	matrixes /multiply .....	vec_\$imat_mult16
multiply two 32-bit integer	matrixes vec_\$imat_multn: .....	vec_\$imat_multn
multiply two 16-bit integer	matrixes vec_\$imat_multn16: .....	vec_\$imat_multn16
and add 16-bit vectors in	matrixes /scale .....	vec_\$imult_add16_i
and add 32-bit vectors in	matrixes /scale .....	vec_\$imult_add_i
16-bit integer vectors in	matrixes /subtract .....	vec_\$isub16_i
32-bit integer vectors in	matrixes /subtract .....	vec_\$isub_i
two 4x4 single-precision	matrixes /multiply .....	vec_\$imat_mult
multiply two single-precision	matrixes vec_\$imat_multn: .....	vec_\$imat_multn
single-precision vectors in	matrixes /scale and add .....	vec_\$imult_add_i
a double-precision vector in	matrixes /vector to .....	vec_\$sp_dp_i
single-precision vectors in	matrixes vec_\$sub_1: subtract .....	vec_\$sub_i
16-bit/ vec_\$imax16: find the	maximum absolute value in a .....	vec_\$imax16
32-bit/ vec_\$imax: find the	maximum absolute value in a .....	vec_\$imax
vec_\$dmax: find the	maximum absolute value in a/ .....	vec_\$dmax
vec_\$max: find the	maximum absolute value in a/ .....	vec_\$max
vec_\$imax16_i: find the	maximum absolute value in a/ .....	vec_\$imax16_i
vector/ vec_\$imax_i: find the	maximum absolute value in a .....	vec_\$imax_i
vector/ vec_\$dmax_i: find the	maximum absolute value in a .....	vec_\$dmax_i
vector/ vec_\$max_i: find the	maximum absolute value in a .....	vec_\$max_i
client buffer size	mbx_\$client_window: return .....	mbx_\$client_window
mailbox	mbx_\$close: close a channel or .....	mbx_\$close
channel without waiting	mbx_\$cond_get_rec_chan: read a .....	mbx_\$cond_get_rec_chan
read a set of channels/	mbx_\$cond_get_rec_chan_set: .....	mbx_\$cond_get_rec_chan_set
mailbox	mbx_\$create_server: create a .....	mbx_\$create_server
channel	mbx_\$deallocate: deallocate a .....	mbx_\$deallocate
message without waiting	mbx_\$get_conditional: read a .....	mbx_\$get_conditional
a mailbox eventcount	mbx_\$get_ec: get a pointer to .....	mbx_\$get_ec
from a mailbox	mbx_\$get_rec: get a message .....	mbx_\$get_rec
mailbox channel	mbx_\$get_rec_chan: read a .....	mbx_\$get_rec_chan

set of channels	mbx_\$get_rec_chan_set: read a .....	mbx_\$get_rec_chan_set
	mbx_\$intro: using mailboxes .....	mbx_\$intro
channel in a mailbox	mbx_\$open: open a client .....	mbx_\$open
message	mbx_\$put_chr: send a partial .....	mbx_\$put_chr
partial message without/	mbx_\$put_chr_cond: send a .....	mbx_\$put_chr_cond
	mbx_\$put_rec: send a message .....	mbx_\$put_rec
message without waiting	mbx_\$put_rec_cond: send a .....	mbx_\$put_rec_cond
server buffer size	mbx_\$server_window: return .....	mbx_\$server_window
mailbox helper	mbx_\$start_helper: start a .....	mbx_\$start_helper
channel with a timeout	mbx_\$timed_open: open a client .....	mbx_\$timed_open
error_\$print: print an error	message .....	error_\$print
print a standard error	message error_\$std_format: .....	error_\$std_format
mbx_\$get_rec: get a	message from a mailbox .....	mbx_\$get_rec
/print an error	message in standard format .....	error_\$print_format
mbx_\$put_chr: send a partial	message .....	mbx_\$put_chr
mbx_\$put_rec: send a	message .....	mbx_\$put_rec
mbx_\$get_conditional: read a	message without waiting .....	mbx_\$get_conditional
/send a partial	message without waiting .....	mbx_\$put_chr_cond
mbx_\$put_rec_cond: send a	message without waiting .....	mbx_\$put_rec_cond
gmf_\$intro: using graphics	metafiles .....	gmf_\$intro
put a transcript pad in cooked	mode pad_\$cooked: .....	pad_\$cooked
pad_\$raw: place a pad in raw	mode .....	pad_\$raw
the locator origin in relative	mode /re-establish .....	tpad_\$set_cursor
control/ fpp_\$control:	modify the floating-point .....	fpp_\$control
status register fpp_\$status:	modify the floating-point .....	fpp_\$status
error_\$module: return the	module code from a completion/ .....	error_\$module
loader_\$cond_load: ensure a	module is loaded .....	loader_\$cond_load
loader_\$load: load a	module .....	loader_\$load
pad_\$move:	move the output cursor .....	pad_\$move
wait until an icon expands or	moves pad_\$icon_wait: .....	pad_\$icon_wait
object	ms_\$addmap: map more of a .....	ms_\$addmap
on mapped object access	ms_\$advise: advise the system .....	ms_\$advise
attributes of a mapped object	ms_\$attributes: get the file .....	ms_\$attributes
lock an object	ms_\$scrmapl: create, map, and .....	ms_\$scrmapl
lock a temporary object	ms_\$scremp: create, map, and .....	ms_\$scremp
mapped object with the file/	ms_\$fw_file: synchronize a .....	ms_\$fw_file
synchronize a mapped object/	ms_\$fw_partial: partially .....	ms_\$fw_partial
objects	ms_\$intro: mapping file system .....	ms_\$intro
object	ms_\$mapl: map and lock an .....	ms_\$mapl
map an object via its XOID	ms_\$mapl_stream: permanently .....	ms_\$mapl_stream
temporary object permanent	ms_\$smk_permanent: make a .....	ms_\$smk_permanent
permanent object temporary	ms_\$smk_temporary: make a .....	ms_\$smk_temporary
mapped objects are on the/	ms_\$neighbors: find out if two .....	ms_\$neighbors
an object	ms_\$relock: change the lock on .....	ms_\$relock
portion of a mapped object	ms_\$remap: map a different .....	ms_\$remap
mapped object	ms_\$truncate: truncate a .....	ms_\$truncate
	ms_\$unmap: unmap an object .....	ms_\$unmap
magtape descriptor file	mts_\$close_desc: close a .....	mts_\$close_desc
descriptor file	mts_\$copy_desc: copy a magtape .....	mts_\$copy_desc
create a default magtape/	mts_\$create_default_desc: .....	mts_\$create_default_desc
descriptor attribute	mts_\$get_attr: get a magtape .....	mts_\$get_attr

descriptor files	mts_\$intro: using magtape .....	mts_\$intro
	mts_\$label: label a magtape .....	mts_\$label
descriptor file	mts_\$open_desc: open a magtape .....	mts_\$open_desc
magtape descriptor file	mts_\$open_desc_uid: open a .....	mts_\$open_desc_uid
descriptor attribute	mts_\$set_attr: set a magtape .....	mts_\$set_attr
vector by a/	vec_\$ipostmult16: multiply a 16-bit integer .....	vec_\$ipostmult16
vector by/	vec_\$ipostmultn16: multiply a 16-bit integer .....	vec_\$ipostmultn16
vector by a/	vec_\$ipremultn16: multiply a 16-bit integer .....	vec_\$ipremultn16
vector/	vec_\$imult_constant16: multiply a 16-bit integer .....	vec_\$imult_constant16
4x4/	vec_\$ipremult16: multiply a 16-bit vector by a .....	vec_\$ipremult16
vector by a/	vec_\$ipostmult: multiply a 32-bit integer .....	vec_\$ipostmult
vector by a/	vec_\$ipostmultn: multiply a 32-bit integer .....	vec_\$ipostmultn
vector by a/	vec_\$ipremultn: multiply a 32-bit integer .....	vec_\$ipremultn
vector/	vec_\$imult_constant: multiply a 32-bit integer .....	vec_\$imult_constant
4x4 matrix	vec_\$ipremult: multiply a 32-bit vector by a .....	vec_\$ipremult
vector by a/	vec_\$dpostmult: multiply a double-precision .....	vec_\$dpostmult
vector by a/	vec_\$dpremult: multiply a double-precision .....	vec_\$dpremult
vector by a/	vec_\$dpostmultn: multiply a double-precision .....	vec_\$dpostmultn
vector by a/	vec_\$dpremultn: multiply a double-precision .....	vec_\$dpremultn
vector/	vec_\$dmult_constant: multiply a double-precision .....	vec_\$dmult_constant
vector by a/	vec_\$spostmult: multiply a single-precision .....	vec_\$spostmult
vector by a/	vec_\$spremult: multiply a single-precision .....	vec_\$spremult
vector by a/	vec_\$spostmultn: multiply a single-precision .....	vec_\$spostmultn
vector by a/	vec_\$spremultn: multiply a single-precision .....	vec_\$spremultn
vector by/	vec_\$mult_constant: multiply a single-precision .....	vec_\$mult_constant
vec_\$imult_constant16_i:	multiply a vector in a 16-bit/ .....	vec_\$imult_constant16_i
vec_\$imult_constant_i:	multiply a vector in a 32-bit/ .....	vec_\$imult_constant_i
vec_\$dmult_constant_i:	multiply a vector in a/ .....	vec_\$dmult_constant_i
vec_\$mult_constant_i:	multiply a vector in a/ .....	vec_\$mult_constant_i
matrixes	vec_\$imat_multn16: multiply two 16-bit integer .....	vec_\$imat_multn16
matrixes	vec_\$imat_multn: multiply two 32-bit integer .....	vec_\$imat_multn
integer/	vec_\$imat_mult16: multiply two 4x4 16-bit .....	vec_\$imat_mult16
integer/	vec_\$imat_mult: multiply two 4x4 32-bit .....	vec_\$imat_mult
vec_\$dmat_mult:	multiply two 4x4/ .....	vec_\$dmat_mult
vec_\$mat_mult:	multiply two 4x4/ .....	vec_\$mat_mult
matrixes	vec_\$dmat_multn: multiply two double-precision .....	vec_\$dmat_multn
matrixes	vec_\$mat_multn: multiply two single-precision .....	vec_\$mat_multn
mutual exclusion lock record	mutex_\$init: initialize a .....	mutex_\$init
exclusive resource locking	mutex_\$intro: mutually .....	mutex_\$intro
exclusion lock	mutex_\$lock: obtain a mutual .....	mutex_\$lock
mutual exclusion lock	mutex_\$unlock: release a .....	mutex_\$unlock
mutex_\$lock: obtain a	mutual exclusion lock .....	mutex_\$lock
mutex_\$unlock: release a	mutual exclusion lock .....	mutex_\$unlock
mutex_\$init: initialize a	mutual exclusion lock record .....	mutex_\$init
locking	mutex_\$intro: mutually exclusive resource .....	mutex_\$intro
	name_\$add_link: create a link .....	name_\$add_link
name	name_\$cname: change a leaf .....	name_\$cname
a directory	name_\$create_directory: create .....	name_\$create_directory
permanent file	name_\$create_file: create a .....	name_\$create_file
a directory	name_\$delete_directory: delete .....	name_\$delete_directory
file	name_\$delete_file: delete a .....	name_\$delete_file

	name_\$drop_link: delete a link .....	name_\$drop_link
a directory entry	name_\$extract_data_lc: extract .....	name_\$extract_data_lc
naming directory	name_\$get_ndir_lc: get the .....	name_\$get_ndir_lc
pathname	name_\$get_path_lc: get a full .....	name_\$get_path_lc
working directory	name_\$get_wdir_lc: get the .....	name_\$get_wdir_lc
	name_\$intro: the Naming Server .....	name_\$intro
directory	name_\$read_dir_lc: read a .....	name_\$read_dir_lc
pathname associated with a/	name_\$read_link_lc: return the .....	name_\$read_link_lc
directory	name_\$set_ndir: set the naming .....	name_\$set_ndir
working directory	name_\$set_wdir: sets the .....	name_\$set_wdir
/get the current working or	naming directory .....	ios_\$get_dir_lc
/change the current working or	naming directory .....	ios_\$set_dir
name_\$get_ndir_lc: get the	naming directory .....	name_\$get_ndir_lc
name_\$set_ndir: set the	naming directory .....	name_\$set_ndir
name_\$intro: the	Naming Server .....	name_\$intro
tone_\$time: make a	noise of a specified duration .....	tone_\$time
tone_\$intro: make a	noise .....	tone_\$intro
advise the system on mapped	object access ms_\$advice: .....	ms_\$advice
ios_\$create: create an	object and open a stream to it .....	ios_\$create
ios_\$truncate: truncate an	object at the current stream/ .....	ios_\$truncate
ios_\$inq_obj_flags: return	object attributes .....	ios_\$inq_obj_flags
ios_\$set_obj_flag: set	object attributes .....	ios_\$set_obj_flag
ios_dir_\$open: open an	object in an open directory .....	ios_dir_\$open
ios_\$delete: delete an	object .....	ios_\$delete
streams are open on the same	object /determine if two .....	ios_\$equal
return the record type of an	object ios_\$inq_rec_type: .....	ios_\$inq_rec_type
get the type of an	object ios_\$inq_type_uid: .....	ios_\$inq_type_uid
ios_\$open: open an existing	object .....	ios_\$open
ios_\$put: write to an	object .....	ios_\$put
seek to the beginning of an	object ios_\$seek_to_bof: .....	ios_\$seek_to_bof
seek to the end of an	object ios_\$seek_to_eof: .....	ios_\$seek_to_eof
ms_\$ddmap: map more of a	object .....	ms_\$ddmap
file attributes of a mapped	object /get the .....	ms_\$attributes
create, map, and lock an	object ms_\$scmapl: .....	ms_\$scmapl
map, and lock a temporary	object ms_\$scrtmp: create, .....	ms_\$scrtmp
ms_\$mapl: map and lock an	object .....	ms_\$mapl
change the lock on an	object ms_\$relock: .....	ms_\$relock
different portion of a mapped	object ms_\$rmap: map a .....	ms_\$rmap
truncate a mapped	object ms_\$truncate: .....	ms_\$truncate
ms_\$unmap: unmap an	object .....	ms_\$unmap
/make a temporary	object permanent .....	ms_\$mk_permanent
/make a permanent	object temporary .....	ms_\$mk_temporary
ios_\$force_write_file: save an	object to permanent storage .....	ios_\$force_write_file
/permanently map an	object via its XOID .....	ms_\$mapl_stream
/synchronize a mapped	object with the file system .....	ms_\$fw_file
/partially synchronize a mapped	object with the file system .....	ms_\$fw_partial
/find out if two mapped	objects are on the same volume .....	ms_\$neighbors
ms_\$intro: mapping file system	objects .....	ms_\$intro
/change an	object's pathname .....	ios_\$change_path_name
ios_\$inq_path_name_lc: get an	object's pathname .....	ios_\$inq_path_name_lc
ios_\$set_rec_type: change an	object's record type .....	ios_\$set_rec_type

mutex_\$lock:	obtain a mutual exclusion lock .....	mutex_\$lock
/get the total display region	occupied by a window .....	pad_\$inq_full_window
cal_\$intro: the	Domain/OS calendar service .....	cal_\$intro
error_\$intro:	Domain/OS error reporting .....	error_\$intro
fault_\$intro:	Domain/OS fault codes .....	fault_\$intro
osinfo_\$intro:	Domain/OS information .....	osinfo_\$intro
Communication ipc_\$intro:	Domain/OS Interprocess .....	ipc_\$intro
loader_\$intro: the	Domain/OS loader .....	loader_\$intro
prf_\$intro: the	Domain/OS print library .....	prf_\$intro
pm_\$intro: the	Domain/OS Process Manager .....	pm_\$intro
osinfo_\$get_rev: get	Domain/OS revision information .....	osinfo_\$get_rev
task_\$intro: the	Domain/OS task library .....	task_\$intro
time_\$intro: the	Domain/OS time service .....	time_\$intro
mailbox mbx_\$open:	open a client channel in a .....	mbx_\$open
timeout mbx_\$timed_open:	open a client channel with a .....	mbx_\$timed_open
gmf_\$open:	open a GMF .....	gmf_\$open
mts_\$open_desc:	open a magtape descriptor file .....	mts_\$open_desc
mts_\$open_desc_uid:	open a magtape descriptor file .....	mts_\$open_desc_uid
/create an object and	open a stream to it .....	ios_\$create
ios_\$open:	open an existing object .....	ios_\$open
pbufs_\$open:	open an existing paste buffer .....	pbufs_\$open
ipc_\$open:	open an IPC socket .....	ipc_\$open
directory ios_dir_\$open:	open an object in an open .....	ios_dir_\$open
open an object in an	open directory ios_dir_\$open: .....	ios_dir_\$open
/test whether a stream is	open on a directory .....	ios_dir_\$isa
/test whether a stream is	open on a local DM pad .....	pad_\$isa_dm_pad
/determine if two streams are	open on the same object .....	ios_\$equal
set a scale factor for cursor	operations pad_\$set_scale: .....	pad_\$set_scale
prf_\$set_option: set an	option in a print request .....	prf_\$set_option
get a print request	option prf_\$inq_option: .....	prf_\$inq_option
prf_\$config_file: set print	options from a file .....	prf_\$config_file
initialize print request	options prf_\$init: .....	prf_\$init
sio_\$control: set serial line	options .....	sio_\$control
sio_\$inquire: get serial line	options .....	sio_\$inquire
/re-establish the locator	origin in relative mode .....	tpad_\$set_cursor
pgm_\$make_orphan:	orphan a process .....	pgm_\$make_orphan
revision information	osinfo_\$get_rev: get Domain/OS .....	osinfo_\$get_rev
information	osinfo_\$intro: Domain/OS .....	osinfo_\$intro
pad_\$inq_position: find the	output cursor .....	pad_\$inq_position
pad_\$move: move the	output cursor .....	pad_\$move
the completion status and	output value /change .....	task_\$set_result
formatted write to standard	output vfmt_\$write: .....	vfmt_\$write
formatted write to standard	output vfmt_\$write10: .....	vfmt_\$write10
formatted write to standard	output vfmt_\$write2: .....	vfmt_\$write2
formatted write to standard	output vfmt_\$write5: .....	vfmt_\$write5
fpp_\$intro: the Floating-Point	Package .....	fpp_\$intro
variable formatting	package vfmt_\$intro: .....	vfmt_\$intro
/create a new	pad and a window to view it .....	pad_\$create_window
pad_\$create_icon: create a new	pad and associated window icon .....	pad_\$create_icon
pad_\$create: create a new	pad and window pane .....	pad_\$create
/close a window when its	pad closes .....	pad_\$set_auto_close

pad_\$cooked: put a transcript	pad in cooked mode .....	pad_\$cooked
pad_\$raw: place a	pad in raw mode .....	pad_\$raw
incomplete record to an input	pad /force an .....	pad_\$force_prompt
list the windows on a	pad pad_\$inq_windows: .....	pad_\$inq_windows
a stream is open on a local DM	pad /test whether .....	pad_\$isa_dm_pad
set tab stops within a	pad pad_\$set_tabs: .....	pad_\$set_tabs
pad_\$set_view: define the	pad region under a window .....	pad_\$set_view
pad_\$inq_view: find out the	pad region under the window .....	pad_\$inq_view
test a stream for the	pad trait pad_\$isa: .....	pad_\$isa
frame	pad_\$clear_frame: clear a .....	pad_\$clear_frame
frame	pad_\$close_frame: close a .....	pad_\$close_frame
pad in cooked mode	pad_\$cooked: put a transcript .....	pad_\$cooked
position reports	pad_\$scr_enable: enable cursor .....	pad_\$scr_enable
and window pane	pad_\$create: create a new pad .....	pad_\$create
frame	pad_\$create_frame: create a .....	pad_\$create_frame
pad and associated window/	pad_\$create_icon: create a new .....	pad_\$create_icon
new pad and a window to view/	pad_\$create_window: create a .....	pad_\$create_window
function key	pad_\$def_ptk: define a program .....	pad_\$def_ptk
delete a frame	pad_\$delete_frame: clear and .....	pad_\$delete_frame
Manager command	pad_\$dm_cmd: execute a Display .....	pad_\$dm_cmd
edit window closes	pad_\$edit_wait: wait until an .....	pad_\$edit_wait
incomplete record to an input/	pad_\$force_prompt: force an .....	pad_\$force_prompt
icon expands or moves	pad_\$icon_wait: wait until an .....	pad_\$icon_wait
the display type	pad_\$inq_disp_type: find out .....	pad_\$inq_disp_type
the current font	pad_\$inq_font: find out about .....	pad_\$inq_font
total display region occupied/	pad_\$inq_full_window: get the .....	pad_\$inq_full_window
information	pad_\$inq_icon: get icon .....	pad_\$inq_icon
current icon font	pad_\$inq_icon_font: get the .....	pad_\$inq_icon_font
the keyboard	pad_\$inq_kbd: find out about .....	pad_\$inq_kbd
output cursor	pad_\$inq_position: find the .....	pad_\$inq_position
pad region under the window	pad_\$inq_view: find out the .....	pad_\$inq_view
windows on a pad	pad_\$inq_windows: list the .....	pad_\$inq_windows
Manager	pad_\$intro: the Display .....	pad_\$intro
the pad trait	pad_\$isa: test a stream for .....	pad_\$isa
a stream is open on a local/	pad_\$isa_dm_pad: test whether .....	pad_\$isa_dm_pad
a window is an icon	pad_\$is_icon: find out whether .....	pad_\$is_icon
character font	pad_\$load_font: load a .....	pad_\$load_font
keyboard cursor	pad_\$locate: locate the .....	pad_\$locate
window into an icon	pad_\$make_icon: change a .....	pad_\$make_icon
window invisible	pad_\$make_invisible: make a .....	pad_\$make_invisible
cursor	pad_\$move: move the output .....	pad_\$move
push a window	pad_\$pop_push_window: pop or .....	pad_\$pop_push_window
mode	pad_\$raw: place a pad in raw .....	pad_\$raw
window visible and full-size	pad_\$select_window: make a .....	pad_\$select_window
window when its pad closes	pad_\$set_auto_close: close a .....	pad_\$set_auto_close
a window border	pad_\$set_border: add or remove .....	pad_\$set_border
establish a window region	pad_\$set_full_window: .....	pad_\$set_full_window
new icon font	pad_\$set_icon_font: choose a .....	pad_\$set_icon_font
an icon position and icon/	pad_\$set_icon_pos: establish .....	pad_\$set_icon_pos
factor for cursor operations	pad_\$set_scale: set a scale .....	pad_\$set_scale
within a pad	pad_\$set_tabs: set tab stops .....	pad_\$set_tabs

region under a window	pad_\$set_view: define the pad .....	pad_\$set_view
loaded font	pad_\$use_font: use a new .....	pad_\$use_font
create a new pad and window	pane pad_\$create: .....	pad_\$create
mbx_\$put_chr: send a	partial message .....	mbx_\$put_chr
mbx_\$put_chr_cond: send a	partial message without/ .....	mbx_\$put_chr_cond
object with/ ms_\$fw_partial:	partially synchronize a mapped .....	ms_\$fw_partial
pbufs_\$create: create a	paste buffer .....	pbufs_\$create
pbufs_\$open: open an existing	paste buffer .....	pbufs_\$open
pbufs_\$intro: using	paste buffers .....	pbufs_\$intro
name_\$read_link_lc: return the	pathname associated with a/ .....	name_\$read_link_lc
/change an object's	pathname .....	ios_\$change_path_name
/get an object's	pathname .....	ios_\$inq_path_name_lc
name_\$get_path_lc: get a full	pathname .....	name_\$get_path_lc
buffer	pbufs_\$create: create a paste .....	pbufs_\$create
buffers	pbufs_\$intro: using paste .....	pbufs_\$intro
paste buffer	pbufs_\$open: open an existing .....	pbufs_\$open
name_\$create_file: create a	permanent file .....	name_\$create_file
make a temporary object	permanent ms_\$mk_permanent: .....	ms_\$mk_permanent
ms_\$mk_temporary: make a	permanent object temporary .....	ms_\$mk_temporary
/save an object to	permanent storage .....	ios_\$force_write_file
its XOID ms_\$mapl_stream:	permanently map an object via .....	ms_\$mapl_stream
clean-up handler	pfm_\$cleanup: establish a .....	pfm_\$cleanup
asynchronous faults	pfm_\$enable: enable .....	pfm_\$enable
fault and save a traceback	pfm_\$error_trap: simulate a .....	pfm_\$error_trap
establish a fault handler	pfm_\$establish_fault_handler: .....	pfm_\$establish_fault_handler
asynchronous faults	pfm_\$inhibit: inhibit .....	pfm_\$inhibit
	pfm_\$intro: managing faults .....	pfm_\$intro
release a fault handler	pfm_\$release_fault_handler: .....	pfm_\$release_fault_handler
clean-up handler	pfm_\$reset_cleanup: reset a .....	pfm_\$reset_cleanup
clean-up handlers	pfm_\$rls_cleanup: release .....	pfm_\$rls_cleanup
calling process	pfm_\$signal: signal the .....	pfm_\$signal
argument	pgm_\$del_arg: delete a program .....	pgm_\$del_arg
	pgm_\$exit: exit a program .....	pgm_\$exit
argument	pgm_\$get_arg: get a program .....	pgm_\$get_arg
argument vector	pgm_\$get_args: get the program .....	pgm_\$get_args
eventcount	pgm_\$get_ec: get a process .....	pgm_\$get_ec
UID	pgm_\$get_puid: get a process .....	pgm_\$get_puid
manager	pgm_\$intro: the program .....	pgm_\$intro
	pgm_\$invoke: invoke a program .....	pgm_\$invoke
process	pgm_\$make_orphan: orphan a .....	pgm_\$make_orphan
process	pgm_\$proc_wait: wait on a .....	pgm_\$proc_wait
exit severity level	pgm_\$set_severity: set the .....	pgm_\$set_severity
ctm_\$inc_use_count: increment	pixel value use counts .....	ctm_\$inc_use_count
ctm_\$release_pv: decrement	pixel value use counts .....	ctm_\$release_pv
ctm_\$alloc_pv: allocate unused	pixel values .....	ctm_\$alloc_pv
ctm_\$mark_read_only: share	pixel values with other/ .....	ctm_\$mark_read_only
gmf_\$copy_plane: copy a bitmap	plane to a GMF .....	gmf_\$copy_plane
/copy part of a bitmap	plane to a GMF .....	gmf_\$copy_subplane
directory	pm_\$get_home_txt: get the home .....	pm_\$get_home_txt
	pm_\$get_sid_txt: get the SID .....	pm_\$get_sid_txt
Manager	pm_\$intro: the Domain/OS Process .....	pm_\$intro

eventcount	mbx_\$get_ec: get a pointer to a mailbox .....	mbx_\$get_ec
	ios_\$get_ec: get a pointer to a stream eventcount .....	ios_\$get_ec
	ios_\$locate: get a pointer to data in a stream .....	ios_\$locate
	time_\$get_ec: get a pointer to the time eventcount .....	time_\$get_ec
	tpad_\$set_mode: set pointing device response/ .....	tpad_\$set_mode
	read/write storage from a pool /allocate .....	rws_\$alloc_rw_pool
	pad_\$pop_push_window: pop or push a window .....	pad_\$pop_push_window
ms_\$remap:	map a different portion of a mapped object .....	ms_\$remap
	/calls to manipulate predefined large sets .....	lib_\$intro
	options from a file prf_\$config_file: set print .....	prf_\$config_file
	job at the current site prf_\$edit_job: edit a print .....	prf_\$edit_job
	of printers at the current/ prf_\$get_printers: get a list .....	prf_\$get_printers
	print sites prf_\$get_sites: get a list of .....	prf_\$get_sites
	request options prf_\$init: initialize print .....	prf_\$init
	request option prf_\$inq_option: get a print .....	prf_\$inq_option
	library prf_\$intro: the Domain/OS print .....	prf_\$intro
	for printing prf_\$name_print: spool a file .....	prf_\$name_print
	request prf_\$queue_file: queue a print .....	prf_\$queue_file
	print jobs in the queue prf_\$read_queue: get a list of .....	prf_\$read_queue
	in a print request prf_\$set_option: set an option .....	prf_\$set_option
	print job prf_\$signal_printer: signal a .....	prf_\$signal_printer
	a stream prf_\$stream_print: print from .....	prf_\$stream_print
	error_\$std_format: print a standard error message .....	error_\$std_format
	error_\$print: print an error message .....	error_\$print
standard/	error_\$print_format: print an error message in .....	error_\$print_format
	error_\$print_name: print error text with a name .....	error_\$print_name
	prf_\$stream_print: print from a stream .....	prf_\$stream_print
	prf_\$edit_job: edit a print job at the current site .....	prf_\$edit_job
	prf_\$signal_printer: signal a print job .....	prf_\$signal_printer
	prf_\$read_queue: get a list of print jobs in the queue .....	prf_\$read_queue
	prf_\$intro: the Domain/OS print library .....	prf_\$intro
	prf_\$config_file: set print options from a file .....	prf_\$config_file
	prf_\$inq_option: get a print request option .....	prf_\$inq_option
	prf_\$init: initialize print request options .....	prf_\$init
	prf_\$queue_file: queue a print request .....	prf_\$queue_file
	/set an option in a print request .....	prf_\$set_option
	prf_\$get_sites: get a list of print sites .....	prf_\$get_sites
	/get a list of printers at the current site .....	prf_\$get_printers
	spool a file for printing prf_\$name_print: .....	prf_\$name_print
	loader_\$inlib: create a privately installed library .....	loader_\$inlib
change the protected subsystem	privilege level acm_\$intro: .....	acm_\$intro
	CPU time proc1_\$get_cput: get elapsed .....	proc1_\$get_cput
Process Manager	proc1_\$intro: the Level 1 .....	proc1_\$intro
process information	proc2_\$get_info: get level 2 .....	proc2_\$get_info
Process Manager	proc2_\$intro: the Level 2 .....	proc2_\$intro
process UIDs	proc2_\$list: list level 2 .....	proc2_\$list
of the calling process	proc2_\$who_am_i: get the UID .....	proc2_\$who_am_i
	pgm_\$get_ec: get a process eventcount .....	pgm_\$get_ec
	proc2_\$get_info: get level 2 process information .....	proc2_\$get_info
	pm_\$intro: the Domain/OS Process Manager .....	pm_\$intro
	proc1_\$intro: the Level 1 Process Manager .....	proc1_\$intro

proc2_Sintro: the Level 2 signal the calling	Process Manager	proc2_Sintro
pgm_Smake_orphan: orphan a process	process pfm_Ssignal:	pfm_Ssignal
pgm_Sproc_wait: wait on a process	process	pgm_Smake_orphan
get the UID of the calling process	proc2_Swho_am_i:	proc2_Swho_am_i
pgm_Sget_puid: get a process UID	process UID	pgm_Sget_puid
proc2_Slist: list level 2 processes	process UIDs	proc2_Slist
/share pixel values with other task_Syield: yield the processor	product of two 16-bit integer/	task_Syield
vec_Sidot16: return the dot product of two 16-bit integer/	product of two 32-bit integer/	vec_Sidot16
vec_Sidot: return the dot product of two/	product of two/	vec_Sidot
vec_Sddot: return the dot product of two/	product of two vectors in/	vec_Sddot
vec_Sdot: return the dot product of two vectors in/	product of two vectors in/	vec_Sdot
vec_Sidot16_i: return the dot product of two vectors in/	product of two vectors in/	vec_Sidot16_i
vec_Sidot_i: return the dot product of two vectors in/	product of two vectors in/	vec_Sidot_i
vec_Sddot_i: return the dot product of two vectors in/	protected subsystem privilege	vec_Sddot_i
vec_Sdot_i: return the dot protected subsystem privilege	push a window	vec_Sdot_i
level aclm_Sintro: change the pad_Spop_push_window: pop or prf_Squeue_file:	queue a print request	aclm_Sintro
a list of print jobs in the touchpad raw data	prf_Sread_queue: get	pad_Spop_push_window
/re-establishes the touchpad raw data range	range /re-establishes	prf_Squeue_file
pad_Sraw: place a pad in raw mode	raw data range	prf_Sread_queue
mbx_Scond_get_rec_chan: read a channel without waiting	raw mode	tpad_Sre_range
ios_dir_Sreaddir: read a directory	read a channel without waiting	tpad_Sre_range
name_Sread_dir_lc: read a directory	read a directory	pad_Sraw
mbx_Sget_rec_chan: read a mailbox channel	read a directory	mbx_Scond_get_rec_chan
mbx_Sget_conditional: read a message without waiting	read a mailbox channel	ios_dir_Sreaddir
mbx_Sget_rec_chan_set: read a set of channels	read a message without waiting	name_Sread_dir_lc
mbx_Scond_get_rec_chan_set: read a set of channels without/	read a set of channels	mbx_Sget_rec_chan
ios_Sget: read data from a stream	read a set of channels without/	mbx_Sget_conditional
vfmt_Srs10: formatted read from a stream	ios_Sget	mbx_Sget_rec_chan_set
vfmt_Srs2: formatted read from a stream	vfmt_Srs10	mbx_Scond_get_rec_chan_set
vfmt_Srs5: formatted read from a stream	vfmt_Srs2	ios_Sget
vfmt_Sdecode10: formatted read from a string	vfmt_Srs5	vfmt_Srs10
vfmt_Sdecode2: formatted read from a string	vfmt_Sdecode10	vfmt_Srs2
vfmt_Sdecode5: formatted read from a string	vfmt_Sdecode2	vfmt_Srs5
vfmt_Sread10: formatted read from standard input	vfmt_Sdecode5	vfmt_Sdecode10
vfmt_Sread2: formatted read from standard input	vfmt_Sread10	vfmt_Sdecode2
vfmt_Sread5: formatted read from standard input	vfmt_Sread2	vfmt_Sdecode5
rws_Salloc_rw_pool: allocate read/write storage from a pool	vfmt_Sread5	vfmt_Sread10
return the length of the next record ios_Sinq_cur_rec_len:	rws_Salloc_rw_pool	vfmt_Sread2
the number of bytes left in a mutual exclusion lock	record ios_Sinq_cur_rec_len	vfmt_Sread5
ios_Sinq_rec_pos: return the record /return	record /return	rws_Salloc_rw_pool
/force an incomplete record /initialize	record /initialize	ios_Sinq_cur_rec_len
/change an object's record position of a stream/	record position of a stream/	ios_Sinq_rec_remainder
ios_Sinq_rec_type: return the record to an input pad	record to an input pad	mutex_Sinit
origin in/ tpad_Sset_cursor: re-establish the locator	record type	ios_Sinq_rec_pos
raw data/ tpad_Sre_range: re-establishes the touchpad	record type of an object	pad_Sforce_prompt
	re-establish the locator	ios_Sset_rec_type
	re-establishes the touchpad	ios_Sinq_rec_type
		tpad_Sset_cursor
		tpad_Sre_range

/get the total display	region occupied by a window	pad_\$inq_full_window
establish a window	region pad_\$set_full_window:	pad_\$set_full_window
pad_\$set_view: define the pad	region under a window	pad_\$set_view
/find out the pad	region under the window	pad_\$set_view
the floating-point control	register fpp_\$control: modify	fpp_\$control
the floating-point status	register fpp_\$status: modify	fpp_\$status
the locator origin in	relative mode /re-establish	tpad_\$set_cursor
pfm_\$release_fault_handler:	release a fault handler	pfm_\$release_fault_handler
lock mutex_\$unlock:	release a mutual exclusion	mutex_\$unlock
task_\$release:	release a task and report	task_\$release
pfm_\$rsls_cleanup:	release clean-up handlers	pfm_\$rsls_cleanup
rws_\$release_heap_pool:	release heap storage	rws_\$release_heap_pool
pad_\$set_border: add or	remove a window border	pad_\$set_border
ios_\$replicate:	replicate a stream	ios_\$replicate
an IPC datagram and await a	reply ipc_\$sar: send	ipc_\$sar
release a task and	report task_\$release:	task_\$release
/initialize standard error	reporting	error_\$init_std_format
error_\$intro: Domain/OS error	reporting	error_\$intro
error_\$find_text: find error	reporting text	error_\$find_text
error_\$get_text: get error	reporting text	error_\$get_text
status_\$intro: status	reporting types and constants	status_\$intro
enable cursor position	reports pad_\$cpr_enable:	pad_\$cpr_enable
prf_\$inq_option: get a print	request option	prf_\$inq_option
prf_\$init: initialize print	request options	prf_\$init
prf_\$queue_file: queue a print	request	prf_\$queue_file
set an option in a print	request prf_\$set_option:	prf_\$set_option
/get the stack size	requirement of an image	loader_\$lookup_stack_size
pfm_\$reset_cleanup:	reset a clean-up handler	pfm_\$reset_cleanup
/mutually exclusive	resource locking	mutex_\$intro
/set pointing device	response characteristics	tpad_\$set_mode
about the current locator	response /get information	tpad_\$inquire
gmf_\$restore_plane:	restore a bitmap from a GMF	gmf_\$restore_plane
ipc_\$srcv:	retrieve an IPC datagram	ipc_\$srcv
ios_\$inq_full_key:	return a full seek key	ios_\$inq_full_key
directory ios_dir_\$telldir:	return a seek key for a	ios_dir_\$telldir
ios_\$inq_short_key:	return a short seek key	ios_\$inq_short_key
ec2_\$read:	return an eventcount value	ec2_\$read
mbx_\$client_window:	return client buffer size	mbx_\$client_window
ios_\$inq_conn_flags:	return connection attributes	ios_\$inq_conn_flags
ios_\$inq_mgr_flags:	return manager attributes	ios_\$inq_mgr_flags
ios_\$inq_obj_flags:	return object attributes	ios_\$inq_obj_flags
mbx_\$server_window:	return server buffer size	mbx_\$server_window
stream/ ios_\$inq_byte_pos:	return the byte position of a	ios_\$inq_byte_pos
16-bit integer/ vec_\$idot16:	return the dot product of two	vec_\$idot16
32-bit integer/ vec_\$idot:	return the dot product of two	vec_\$idot
double-precision/ vec_\$ddot:	return the dot product of two	vec_\$ddot
single-precision/ vec_\$dot:	return the dot product of two	vec_\$dot
vectors in/ vec_\$idot16_i:	return the dot product of two	vec_\$idot16_i
vectors in/ vec_\$idot_i:	return the dot product of two	vec_\$idot_i
vectors in/ vec_\$ddot_i:	return the dot product of two	vec_\$ddot_i
vectors in/ vec_\$dot_i:	return the dot product of two	vec_\$dot_i

completion/ error_ \$code:	return the error code from a .....	error_ \$code
tpad_ \$inq_ \$dtype:	return the last locator used .....	tpad_ \$inq_ \$dtype
record ios_ \$inq_ \$cur_ \$rec_ \$len:	return the length of the next .....	ios_ \$inq_ \$cur_ \$rec_ \$len
completion/ error_ \$module:	return the module code from a .....	error_ \$module
left/ ios_ \$inq_ \$rec_ \$remainder:	return the number of bytes .....	ios_ \$inq_ \$rec_ \$remainder
with a/ name_ \$read_ \$link_ \$lc:	return the pathname associated .....	name_ \$read_ \$link_ \$lc
a stream/ ios_ \$inq_ \$rec_ \$pos:	return the record position of .....	ios_ \$inq_ \$rec_ \$pos
object ios_ \$inq_ \$rec_ \$type:	return the record type of an .....	ios_ \$inq_ \$rec_ \$type
fpp_ \$save_ \$restore_ \$size:	return the size of a/ .....	fpp_ \$save_ \$restore_ \$size
a completion/ error_ \$subsys:	return the subsystem code from .....	error_ \$subsys
task_ \$get_ \$handle:	return the task handle .....	task_ \$get_ \$handle
cal_ \$weekday:	return the weekday for a date .....	cal_ \$weekday
osinfo_ \$get_ \$rev: get Domain/OS	revision information .....	osinfo_ \$get_ \$rev
ios_ \$dir_ \$rewinddir:	rewind a directory .....	ios_ \$dir_ \$rewinddir
deassert subsystem manager	rights aclm_ \$down: .....	aclm_ \$down
assert subsystem manager	rights aclm_ \$up: .....	aclm_ \$up
	rws_ \$alloc: allocate storage .....	rws_ \$alloc
heap storage	rws_ \$alloc_ \$heap_ \$pool: allocate .....	rws_ \$alloc_ \$heap_ \$pool
read/write storage from a/	rws_ \$alloc_ \$rw_ \$pool: allocate .....	rws_ \$alloc_ \$rw_ \$pool
allocation	rws_ \$intro: dynamic storage .....	rws_ \$intro
release heap storage	rws_ \$release_ \$heap_ \$pool: .....	rws_ \$release_ \$heap_ \$pool
/simulate a fault and	save a traceback .....	pfm_ \$error_ \$trap
ios_ \$force_ \$write_ \$file:	save an object to permanent/ .....	ios_ \$force_ \$write_ \$file
new/ fpp_ \$save_ \$restore:	save the current and load a .....	fpp_ \$save_ \$restore
fpp_ \$save:	save the floating-point state .....	fpp_ \$save
vec_ \$iadd_ \$constant16: add a	scalar constant to a 16-bit/ .....	vec_ \$iadd_ \$constant16
vec_ \$iadd_ \$constant: add a	scalar constant to a 32-bit/ .....	vec_ \$iadd_ \$constant
vec_ \$dadd_ \$constant: add a	scalar to a double-precision/ .....	vec_ \$dadd_ \$constant
vec_ \$sadd_ \$constant: add a	scalar to a single-precision/ .....	vec_ \$sadd_ \$constant
vec_ \$iadd_ \$constant16_ \$i: add a	scalar to a vector in a 16-bit/ .....	vec_ \$iadd_ \$constant16_ \$i
vec_ \$iadd_ \$constant_ \$i: add a	scalar to a vector in a 32-bit/ .....	vec_ \$iadd_ \$constant_ \$i
vec_ \$dadd_ \$constant_ \$i: add a	scalar to a vector in a/ .....	vec_ \$dadd_ \$constant_ \$i
vec_ \$sadd_ \$constant_ \$i: add a	scalar to a vector in a/ .....	vec_ \$sadd_ \$constant_ \$i
a double-precision vector by a	scalar /multiply .....	vec_ \$dmult_ \$constant
a double-precision matrix by a	scalar /multiply a vector in .....	vec_ \$dmult_ \$constant_ \$i
a 16-bit integer vector by a	scalar /multiply .....	vec_ \$simult_ \$constant
a 16-bit integer vector by a	scalar /multiply .....	vec_ \$simult_ \$constant16
a 16-bit integer matrix by a	scalar /multiply a vector in .....	vec_ \$simult_ \$constant16_ \$i
a 32-bit integer matrix by a	scalar /multiply a vector in .....	vec_ \$simult_ \$constant_ \$i
a single-precision vector by a	scalar /multiply .....	vec_ \$smult_ \$constant
a single-precision matrix by a	scalar /multiply a vector in .....	vec_ \$smult_ \$constant_ \$i
in/ vec_ \$simult_ \$add16_ \$i:	scale and add 16-bit vectors .....	vec_ \$simult_ \$add16_ \$i
in matrixes vec_ \$simult_ \$add_ \$i:	scale and add 32-bit vectors .....	vec_ \$simult_ \$add_ \$i
vectors in/ vec_ \$dmult_ \$add_ \$i:	scale and add double-precision .....	vec_ \$dmult_ \$add_ \$i
integer/ vec_ \$simult_ \$add16:	scale and add one 16-bit .....	vec_ \$simult_ \$add16
integer/ vec_ \$simult_ \$add:	scale and add one 32-bit .....	vec_ \$simult_ \$add
vec_ \$dmult_ \$add:	scale and add one/ .....	vec_ \$dmult_ \$add
vec_ \$smult_ \$add:	scale and add one/ .....	vec_ \$smult_ \$add
vectors in/ vec_ \$smult_ \$add_ \$i:	scale and add single-precision .....	vec_ \$smult_ \$add_ \$i
pad_ \$set_ \$scale: set a	scale factor for cursor/ .....	pad_ \$set_ \$scale
convert clock ticks into whole	seconds cal_ \$clock_ \$to_ \$sec: .....	cal_ \$clock_ \$to_ \$sec

convert clock ticks into	seconds	cal_\$float_clock:	cal_\$float_clock
cal_\$sec_to_clock: convert	seconds to a clock value	cal_\$sec_to_clock	cal_\$sec_to_clock
/get information on an image	section	loader_\$inquire_section	loader_\$inquire_section
ios_dir_\$seekdir:	seek in a directory	ios_dir_\$seekdir	ios_dir_\$seekdir
ios_dir_\$stelldir: return a	seek key for a directory	ios_dir_\$stelldir	ios_dir_\$stelldir
return a full	seek key	ios_\$inq_full_key:	ios_\$inq_full_key
return a short	seek key	ios_\$inq_short_key:	ios_\$inq_short_key
ios_\$seek:	seek on a stream	ios_\$seek	ios_\$seek
object ios_\$seek_to_bof:	seek to the beginning of an	ios_\$seek_to_bof	ios_\$seek_to_bof
ios_\$seek_to_eof:	seek to the end of an object	ios_\$seek_to_eof	ios_\$seek_to_eof
ios_\$seek_full_key:	seek with a full key	ios_\$seek_full_key	ios_\$seek_full_key
ios_\$seek_short_key:	seek with a short key	ios_\$seek_short_key	ios_\$seek_short_key
mbx_\$put_rec:	send a message	mbx_\$put_rec	mbx_\$put_rec
mbx_\$put_rec_cond:	send a message without waiting	mbx_\$put_rec_cond	mbx_\$put_rec_cond
mbx_\$put_chr:	send a partial message	mbx_\$put_chr	mbx_\$put_chr
waiting mbx_\$put_chr_cond:	send a partial message without	mbx_\$put_chr_cond	mbx_\$put_chr_cond
a reply ipc_\$sar:	send an IPC datagram and await	ipc_\$sar	ipc_\$sar
ipc_\$send:	send an IPC datagram	ipc_\$send	ipc_\$send
sio_\$intro: controlling	serial I/O lines	sio_\$intro	sio_\$intro
sio_\$control: set	serial line options	sio_\$control	sio_\$control
sio_\$inquire: get	serial line options	sio_\$inquire	sio_\$inquire
mbx_\$server_window: return	server buffer size	mbx_\$server_window	mbx_\$server_window
name_\$intro: the Naming	Server	name_\$intro	name_\$intro
to manipulate predefined large	sets lib_\$intro: calls	lib_\$intro	lib_\$intro
name_\$set_wdir:	sets the working directory	name_\$set_wdir	name_\$set_wdir
/set the exit	severity level	pgm_\$set_severity	pgm_\$set_severity
ctm_\$mark_read_only:	share pixel values with other/	ctm_\$mark_read_only	ctm_\$mark_read_only
/seek with a	short key	ios_\$seek_short_key	ios_\$seek_short_key
ios_\$inq_short_key: return a	short seek key	ios_\$inq_short_key	ios_\$inq_short_key
pm_\$get_sid_txt: get the	SID	pm_\$get_sid_txt	pm_\$get_sid_txt
prf_\$signal_printer:	signal a print job	prf_\$signal_printer	prf_\$signal_printer
task_\$signal:	signal a task	task_\$signal	task_\$signal
pfm_\$signal:	signal the calling process	pfm_\$signal	pfm_\$signal
traceback pfm_\$error_trap:	simulate a fault and save a	pfm_\$error_trap	pfm_\$error_trap
scalar /multiply a vector in a	single-precision matrix by a	vec_\$mult_constant_i	vec_\$mult_constant_i
/copy a vector from one	single-precision matrix to/	vec_\$copy_i	vec_\$copy_i
/add a scalar to a vector in a	single-precision matrix	vec_\$add_constant_i	vec_\$add_constant_i
double-precision matrix into a	single-precision matrix /a	vec_\$dp_sp_i	vec_\$dp_sp_i
/value in a vector from a	single-precision matrix	vec_\$max_i	vec_\$max_i
the elements of a vector in a	single-precision matrix /sum	vec_\$sum_i	vec_\$sum_i
/swap two vectors in a	single-precision matrix	vec_\$swap_i	vec_\$swap_i
/zero a vector in a	single-precision matrix	vec_\$zero_i	vec_\$zero_i
/add vectors in two	single-precision matrixes	vec_\$add_vector_i	vec_\$add_vector_i
dot product of two vectors in	single-precision matrixes /the	vec_\$dot_i	vec_\$dot_i
/multiply two 4x4	single-precision matrixes	vec_\$mat_mult	vec_\$mat_mult
vec_\$mat_multn: multiply two	single-precision matrixes	vec_\$mat_multn	vec_\$mat_multn
vec_\$postmult: multiply a	single-precision vector by a/	vec_\$postmult	vec_\$postmult
vec_\$premult: multiply a	single-precision vector by a/	vec_\$premult	vec_\$premult
vec_\$postmultn: multiply a	single-precision vector by a/	vec_\$postmultn	vec_\$postmultn
vec_\$premultn: multiply a	single-precision vector by a/	vec_\$premultn	vec_\$premultn
vec_\$mult_constant: multiply a	single-precision vector by a/	vec_\$mult_constant	vec_\$mult_constant

vec_\$sp_dp_i: copy a	single-precision vector to a/	vec_\$sp_dp_i
vec_\$sp_dp: copy a	single-precision vector to a/	vec_\$sp_dp
another /scale and add one	single-precision vector to	vec_\$mult_add
/add a scalar to a	single-precision vector	vec_\$add_constant
vec_\$copy: copy a	single-precision vector	vec_\$copy
a double-precision vector to a	single-precision vector /copy	vec_\$dp_sp
vec_\$sinit: initialize a	single-precision vector	vec_\$sinit
maximum absolute value in a	single-precision vector /the	vec_\$smax
/sum the elements of a	single-precision vector	vec_\$sum
vec_\$zero: zero a	single-precision vector	vec_\$zero
vec_\$mult_add_i: scale and add	single-precision vectors in/	vec_\$mult_add_i
matrixes vec_\$sub_i: subtract	single-precision vectors in	vec_\$sub_i
vec_\$add_vector: add two	single-precision vectors	vec_\$add_vector
/return the dot product of two	single-precision vectors	vec_\$dot
vec_\$sub: subtract	single-precision vectors	vec_\$sub
vec_\$swap: swap two	single-precision vectors	vec_\$swap
options	sio_\$scontrol: set serial line	sio_\$scontrol
options	sio_\$sinquire: get serial line	sio_\$sinquire
I/O lines	sio_\$sintro: controlling serial	sio_\$sintro
a print job at the current	site prf_\$edit_job: edit	prf_\$edit_job
of printers at the current	site /get a list	prf_\$get_printers
get a list of print	sites prf_\$get_sites:	prf_\$get_sites
/set the locate buffer	size	ios_\$set_locate_buffer_size
return client buffer	size mbx_\$client_window:	mbx_\$client_window
return server buffer	size mbx_\$server_window:	mbx_\$server_window
/return the	size of a floating-point state	fpp_\$save_restore_size
/get the stack	size requirement of an image	loader_\$lookup_stack_size
ipc_\$resolve: get an IPC	socket handle	ipc_\$resolve
ipc_\$close: close an IPC	socket	ipc_\$close
ipc_\$create: create an IPC	socket	ipc_\$create
ipc_\$delete: delete an IPC	socket	ipc_\$delete
ipc_\$open: open an IPC	socket	ipc_\$open
tone_\$time: make a noise of a	specified duration	tone_\$time
prf_\$name_print:	spool a file for printing	prf_\$name_print
image /get the	stack size requirement of an	loader_\$lookup_stack_size
error_\$std_format: print a	standard error message	error_\$std_format
/initialize	standard error reporting	error_\$sinit_std_format
/print an error message in	standard format	error_\$sprint_format
formatted read from	standard input vfmt_\$read10:	vfmt_\$read10
formatted read from	standard input vfmt_\$read2:	vfmt_\$read2
formatted read from	standard input vfmt_\$read5:	vfmt_\$read5
formatted write to	standard output vfmt_\$write:	vfmt_\$write
/formatted write to	standard output	vfmt_\$write10
formatted write to	standard output vfmt_\$write2:	vfmt_\$write2
formatted write to	standard output vfmt_\$write5:	vfmt_\$write5
mbx_\$start_helper:	start a mailbox helper	mbx_\$start_helper
/get the	start address of an image	loader_\$lookup_start_addr
/change the completion	status and output value	task_\$set_result
error code from a completion	status /return the	error_\$code
the fail bit in a completion	status error_\$fail: test	error_\$fail
module code from a completion	status /return the	error_\$module

code from a completion	status /return the subsystem	error_\$subsys
modify the floating-point	status register fpp_\$status:	fpp_\$status
constants status_\$intro:	status reporting types and	status_\$intro
reporting types and constants	status_\$intro: status	status_\$intro
pad_\$set_tabs: set tab	stops within a pad	pad_\$set_tabs
rws_\$intro: dynamic	storage allocation	rws_\$intro
/allocate read/write	storage from a pool	rws_\$alloc_rw_pool
/save an object to permanent	storage	ios_\$force_write_file
rws_\$alloc: allocate	storage	rws_\$alloc
allocate heap	storage rws_\$alloc_heap_pool:	rws_\$alloc_heap_pool
/release heap	storage	rws_\$release_heap_pool
ios_dir_\$intro:	stream access to directories	ios_dir_\$intro
/get a pointer to a	stream eventcount	ios_\$get_ec
pad_\$isa: test a	stream for the pad trait	pad_\$isa
ios_\$get_handle: convert a	stream ID to a handle	ios_\$get_handle
ios_\$close: close a	stream	ios_\$close
ios_\$dup: duplicate a	stream	ios_\$dup
ios_\$get: read data from a	stream	ios_\$get
get a pointer to data in a	stream ios_\$locate:	ios_\$locate
ios_\$replicate: replicate a	stream	ios_\$replicate
ios_\$seek: seek on a	stream	ios_\$seek
ios_dir_\$isa: test whether a	stream is open on a directory	ios_dir_\$isa
pad /test whether a	stream is open on a local DM	pad_\$isa_dm_pad
/return the byte position of a	stream marker	ios_\$inq_byte_pos
the record position of a	stream marker /return	ios_\$inq_rec_pos
an object at the current	stream marker /truncate	ios_\$truncate
print from a	stream prf_\$stream_print:	prf_\$stream_print
ios_\$switch: switch a	stream to another ID	ios_\$switch
create an object and open a	stream to it ios_\$create:	ios_\$create
formatted read from a	stream vfmt_\$rs10:	vfmt_\$rs10
formatted read from a	stream vfmt_\$rs2:	vfmt_\$rs2
formatted read from a	stream vfmt_\$rs5:	vfmt_\$rs5
vfmt_\$ws: formatted write to a	stream	vfmt_\$ws
formatted write to a	stream vfmt_\$ws10:	vfmt_\$ws10
formatted write to a	stream vfmt_\$ws2:	vfmt_\$ws2
formatted write to a	stream vfmt_\$ws5:	vfmt_\$ws5
ios_\$equal: determine if two	streams are open on the same/	ios_\$equal
formatted read from a	string vfmt_\$decode10:	vfmt_\$decode10
formatted read from a	string vfmt_\$decode2:	vfmt_\$decode2
formatted read from a	string vfmt_\$decode5:	vfmt_\$decode5
formatted write to a	string vfmt_\$encode:	vfmt_\$encode
formatted write to a	string vfmt_\$encode10:	vfmt_\$encode10
formatted write to a	string vfmt_\$encode2:	vfmt_\$encode2
formatted write to a	string vfmt_\$encode5:	vfmt_\$encode5
error_\$subsys: return the	subsystem code from a/	error_\$subsys
ac1m_\$down: deassert	subsystem manager rights	ac1m_\$down
ac1m_\$up: assert	subsystem manager rights	ac1m_\$up
/change the protected	subsystem privilege level	ac1m_\$intro
vectors in/ vec_\$sub16_i:	subtract 16-bit integer	vec_\$sub16_i
vectors vec_\$sub16:	subtract 16-bit integer	vec_\$sub16
vectors in/ vec_\$sub_i:	subtract 32-bit integer	vec_\$sub_i

vectors vec_\$\$sub:	subtract 32-bit integer .....	vec_\$\$sub
vectors in/ vec_\$\$sub_i:	subtract double-precision .....	vec_\$\$sub_i
vectors vec_\$\$sub:	subtract double-precision .....	vec_\$\$sub
vectors in/ vec_\$\$sub_i:	subtract single-precision .....	vec_\$\$sub_i
vectors vec_\$\$sub:	subtract single-precision .....	vec_\$\$sub
cal_\$\$sub_clock:	subtract two clock values .....	cal_\$\$sub_clock
integer vector vec_\$\$sum16:	sum the elements of a 16-bit .....	vec_\$\$sum16
integer vector vec_\$\$sum:	sum the elements of a 32-bit .....	vec_\$\$sum
double-precision/ vec_\$\$sum:	sum the elements of a .....	vec_\$\$sum
single-precision/ vec_\$\$sum:	sum the elements of a .....	vec_\$\$sum
in a 16-bit/ vec_\$\$sum16_i:	sum the elements of a vector .....	vec_\$\$sum16_i
in a 32-bit/ vec_\$\$sum_i:	sum the elements of a vector .....	vec_\$\$sum_i
in a/ vec_\$\$sum_i:	sum the elements of a vector .....	vec_\$\$sum_i
in a/ vec_\$\$sum_i:	sum the elements of a vector .....	vec_\$\$sum_i
/get a load history	summation .....	loader_\$\$inquire_stat
vectors vec_\$\$swap16:	swap two 16-bit integer .....	vec_\$\$swap16
vectors vec_\$\$swap:	swap two 32-bit integer .....	vec_\$\$swap
vectors vec_\$\$swap:	swap two double-precision .....	vec_\$\$swap
vectors vec_\$\$swap:	swap two single-precision .....	vec_\$\$swap
integer/ vec_\$\$swap16_i:	swap two vectors in a 16-bit .....	vec_\$\$swap16_i
integer matrix vec_\$\$swap_i:	swap two vectors in a 32-bit .....	vec_\$\$swap_i
vec_\$\$swap_i:	swap two vectors in a/ .....	vec_\$\$swap_i
single-precision/ vec_\$\$swap_i:	swap two vectors in a .....	vec_\$\$swap_i
ios_\$\$switch:	switch a stream to another ID .....	ios_\$\$switch
loader_\$\$kg_lookup:	look up a symbol .....	loader_\$\$kg_lookup
with the file/ ms_\$\$fw_file:	synchronize a mapped object .....	ms_\$\$fw_file
ms_\$\$fw_partial:	partially synchronize a mapped object/ .....	ms_\$\$fw_partial
pad_\$\$set_tabs:	set tab stops within a pad .....	pad_\$\$set_tabs
/add to the Known Global Table (KGT)	.....	loader_\$\$kg_define
ctm_\$\$intro: Color Table Manager	.....	ctm_\$\$intro
task_\$\$release: release a task and report	.....	task_\$\$release
task_\$\$get_handle: return the task handle	.....	task_\$\$get_handle
task_\$\$intro: the Domain/OS task library	.....	task_\$\$intro
task_\$\$create: create a task	.....	task_\$\$create
task_\$\$exit: exit a task	.....	task_\$\$exit
get information about a task	task_\$\$get_info: .....	task_\$\$get_info
task_\$\$set_name: name a task	.....	task_\$\$set_name
task_\$\$signal: signal a task	.....	task_\$\$signal
task_\$\$blast: kill a task without cleanup	.....	task_\$\$blast
task_\$\$blast: kill a task	.....	task_\$\$blast
task_\$\$create: create a task	.....	task_\$\$create
task_\$\$exit: exit a task	.....	task_\$\$exit
eventcount	task_\$\$get_ec: get a completion .....	task_\$\$get_ec
task handle	task_\$\$get_handle: return the .....	task_\$\$get_handle
information about a task	task_\$\$get_info: get .....	task_\$\$get_info
/determine whether	tasking is enabled .....	task_\$\$tasking_enabled
library	task_\$\$intro: the Domain/OS task .....	task_\$\$intro
and report	task_\$\$release: release a task .....	task_\$\$release
completion status and output/	task_\$\$set_name: name a task .....	task_\$\$set_name
	task_\$\$set_result: change the .....	task_\$\$set_result
	task_\$\$signal: signal a task .....	task_\$\$signal

determine whether tasking is/processor	task_Stacking_enabled: .....	task_Stacking_enabled
make a permanent object	task_Syield: yield the .....	task_Syield
create, map, and lock	temporary ms_Smk_temporary: .....	ms_Smk_temporary
ms_Smk_permanent: make a	temporary object ms_Scrtmp: .....	ms_Scrtmp
trait pad_Sisa:	temporary object permanent .....	ms_Smk_permanent
completion/ error_Sfail:	test a stream for the pad .....	pad_Sisa
on a directory ios_dir_Sisa:	test the fail bit in a .....	error_Sfail
on a local/ pad_Sisa_dm_pad:	test whether a stream is open .....	ios_dir_Sisa
find error reporting	test whether a stream is open .....	pad_Sisa_dm_pad
get error reporting	text error_Sfind_text: .....	error_Sfind_text
error_Sprint_name: print error	text error_Sget_text: .....	error_Sget_text
/convert clock	text with a name .....	error_Sprint_name
/convert clock	ticks into seconds .....	cal_Sfloat_clock
decode an ascii	ticks into whole seconds .....	cal_Sclock_to_sec
get the local date and	time cal_Sdecode_ascii_time: .....	cal_Sdecode_ascii_time
encode a date and	time cal_Sdecode_local_time: .....	cal_Sdecode_local_time
get the local	time cal_Sencode_time: .....	cal_Sencode_time
get a pointer to the	time cal_Sget_local_time: .....	cal_Sget_local_time
get elapsed CPU	time eventcount time_Sget_ec: .....	time_Sget_ec
time_Sintro: the Domain/OS	time procl_Sget_cput: .....	procl_Sget_cput
/convert local	time service .....	time_Sintro
/decode an ASCII	time to UTC .....	cal_Sremove_local_offset
cal_Sget_info: get the local	time zone .....	cal_Sdecode_ascii_tzdiff
/set the local	time zone .....	cal_Sget_info
clock value	time zone .....	cal_Swrite_timezone
the time eventcount	time_Sclock: get the system .....	time_Sclock
service	time_Sget_ec: get a pointer to .....	time_Sget_ec
open a client channel with a	time_Sintro: the Domain/OS time .....	time_Sintro
interval	timeout mbx_Stimed_open: .....	mbx_Stimed_open
specified duration	time_Swait: wait for an .....	time_Swait
pad_Sinq_full_window: get the	tone_Sintro: make a noise .....	tone_Sintro
tpad_Sintro: locator	tone_Stime: make a noise of a .....	tone_Stime
/re-establishes the	total display region occupied/ .....	pad_Sinq_full_window
last locator used	(touchpad) manager calls .....	tpad_Sintro
about the current locator/	touchpad raw data range .....	tpad_Sre_range
(touchpad) manager calls	tpad_Sinq_dtype: return the .....	tpad_Sinq_dtype
the touchpad raw data range	tpad_Sinquire: get information .....	tpad_Sinquire
the locator origin in/	tpad_Sintro: locator .....	tpad_Sintro
device response/	tpad_Sre_range: re-establishes .....	tpad_Sre_range
simulate a fault and save a	tpad_Sset_cursor: re-establish .....	tpad_Sset_cursor
test a stream for the pad	tpad_Sset_mode: set pointing .....	tpad_Sset_mode
pad_Scooked: put a	traceback pfm_Serror_trap: .....	pfm_Serror_trap
/wait for an eventcount	trait pad_Sisa: .....	pad_Sisa
wait for an eventcount	transcript pad in cooked mode .....	pad_Scooked
/wait for an eventcount	trigger during I/O .....	ec2_Swait_slow_io
ms_Struncate:	trigger ec2_Swait: .....	ec2_Swait
current stream/ ios_Struncate:	trigger or asynchronous fault .....	ec2_Swait_svc
change an object's record	truncate a mapped object .....	ms_Struncate
/return the record	truncate an object at the .....	ios_Struncate
	type ios_Sset_rec_type: .....	ios_Sset_rec_type
	type of an object .....	ios_Sinq_rec_type

ios_Sinq_type_uid: get the	type of an object	ios_Sinq_type_uid
find out the display	type pad_Sinq_disp_type:	pad_Sinq_disp_type
/status reporting	types and constants	status_Sintro
proc2_\$who_am_i: get the	UID of the calling process	proc2_\$who_am_i
pgm_\$get_puid: get a process	UID	pgm_\$get_puid
list level 2 process	UIDs proc2_\$list:	proc2_\$list
ms_Sunmap:	unmap an object	ms_Sunmap
ctm_Salloc_pv: allocate	unused pixel values	ctm_Salloc_pv
gmf_Sintro:	using graphics metafiles	gmf_Sintro
mts_Sintro:	using magtape descriptor files	mts_Sintro
mbx_Sintro:	using mailboxes	mbx_Sintro
pbufs_Sintro:	using paste buffers	pbufs_Sintro
convert local time to	UTC cal_Sremove_local_offset:	cal_Sremove_local_offset
/convert a	UTC clock value to a local one	cal_Sapply_local_offset
decode a clock	value cal_Sdecode_time:	cal_Sdecode_time
convert seconds to a clock	value cal_Ssec_to_clock:	cal_Ssec_to_clock
return an eventcount	value ec2_Sread:	ec2_Sread
/find the maximum absolute	value in a 16-bit integer/	vec_Simax16
/find the maximum absolute	value in a 32-bit integer/	vec_Simax
/find the maximum absolute	value in a double-precision/	vec_Sdmax
/find the maximum absolute	value in a single-precision/	vec_Ssmax
/find the maximum absolute	value in a vector from a/	vec_Simax16_i
/find the maximum absolute	value in a vector from a/	vec_Simax_i
/find the maximum absolute	value in a vector from a/	vec_Sdmax_i
/find the maximum absolute	value in a vector from a/	vec_Ssmax_i
completion status and output	value /change the	task_Sset_result
get the system clock	value time_Sclock:	time_Sclock
/convert a UTC clock	value to a local one	cal_Sapply_local_offset
/increment pixel	value use counts	ctm_Sinc_use_count
/decrement pixel	value use counts	ctm_Srelease_pv
cal_\$add_clock: add two clock	values	cal_\$add_clock
compare two clock	values cal_\$cmp_clock:	cal_\$cmp_clock
subtract two clock	values cal_\$sub_clock:	cal_\$sub_clock
allocate unused pixel	values ctm_Salloc_pv:	ctm_Salloc_pv
/share pixel	values with other processes	ctm_Smark_read_only
vfmt_Sintro:	variable formatting package	vfmt_Sintro
scalar to a single-precision/	vec_\$add_constant: add a	vec_\$add_constant
scalar to a vector in a/	vec_\$add_constant_i: add a	vec_\$add_constant_i
single-precision vectors	vec_\$add_vector: add two	vec_\$add_vector
in two single-precision/	vec_\$add_vector_i: add vectors	vec_\$add_vector_i
single-precision vector	vec_\$copy: copy a	vec_\$copy
from one single-precision/	vec_\$copy_i: copy a vector	vec_\$copy_i
scalar to a double-precision/	vec_\$dadd_constant: add a	vec_\$dadd_constant
scalar to a vector in a/	vec_\$dadd_constant_i: add a	vec_\$dadd_constant_i
double-precision vectors	vec_\$dadd_vector: add two	vec_\$dadd_vector
vectors in two/	vec_\$dadd_vector_i: add	vec_\$dadd_vector_i
double-precision vector	vec_\$dcopy: copy a	vec_\$dcopy
from one double-precision/	vec_\$dcopy_i: copy a vector	vec_\$dcopy_i
product of two/	vec_\$ddot: return the dot	vec_\$ddot
product of two vectors in/	vec_\$ddot_i: return the dot	vec_\$ddot_i
double-precision vector	vec_\$dinit: initialize a	vec_\$dinit

4x4 double-precision/ double-precision matrixes	vec_\$\$mat_mult: multiply two .....	vec_\$\$mat_mult
absolute value in a/ absolute value in a vector/ one double-precision vector/ add double-precision vectors/ a double-precision vector by/ multiply a vector in a/ product of two/ product of two vectors in/ double-precision vector by a/ double-precision vector by a/ double-precision vector by a/ double-precision vector by a/ double-precision vector to a/ from a double-precision/ double-precision vectors double-precision vectors in/ a double-precision vector of a vector in a/ double-precision vectors in a double-precision matrix double-precision vector a double-precision matrix scalar constant to a 32-bit/ scalar constant to a 16-bit/ scalar to a vector in a/ scalar to a vector in a/ 32-bit integer vectors 16-bit integer vectors vectors in two 16-bit integer/ vectors in two 32-bit integer/ integer vector integer vector from one 16-bit integer/ from one 32-bit integer/ product of two 32-bit integer/ product of two 16-bit integer/ product of two vectors in/ product of two vectors in/ 32-bit integer vector 16-bit integer vector 4x4 32-bit integer/ 4x4 16-bit integer/ 32-bit integer matrixes two 16-bit integer matrixes absolute value in a 32-bit/ absolute value in a 16-bit/ maximum absolute value in a/ absolute value in a vector/ one 32-bit integer vector to/	vec_\$\$mat_multn: multiply two .....	vec_\$\$mat_multn
	vec_\$\$max: find the maximum .....	vec_\$\$max
	vec_\$\$max_i: find the maximum .....	vec_\$\$max_i
	vec_\$\$mult_add: scale and add .....	vec_\$\$mult_add
	vec_\$\$mult_add_i: scale and .....	vec_\$\$mult_add_i
	vec_\$\$mult_constant: multiply .....	vec_\$\$mult_constant
	vec_\$\$mult_constant_i: .....	vec_\$\$mult_constant_i
	vec_\$\$dot: return the dot .....	vec_\$\$dot
	vec_\$\$dot_i: return the dot .....	vec_\$\$dot_i
	vec_\$\$postmult: multiply a .....	vec_\$\$postmult
	vec_\$\$postmultn: multiply a .....	vec_\$\$postmultn
	vec_\$\$premult: multiply a .....	vec_\$\$premult
	vec_\$\$premultn: multiply a .....	vec_\$\$premultn
	vec_\$\$p_sp: copy a .....	vec_\$\$p_sp
	vec_\$\$p_sp_i: copy a vector .....	vec_\$\$p_sp_i
	vec_\$\$sub: subtract .....	vec_\$\$sub
	vec_\$\$sub_i: subtract .....	vec_\$\$sub_i
	vec_\$\$sum: sum the elements of .....	vec_\$\$sum
	vec_\$\$sum_i: sum the elements .....	vec_\$\$sum_i
	vec_\$\$swap: swap two .....	vec_\$\$swap
	vec_\$\$swap_i: swap two vectors .....	vec_\$\$swap_i
	vec_\$\$zero: zero a .....	vec_\$\$zero
	vec_\$\$zero_i: zero a vector in .....	vec_\$\$zero_i
	vec_\$\$iadd_constant: add a .....	vec_\$\$iadd_constant
	vec_\$\$iadd_constant16: add a .....	vec_\$\$iadd_constant16
	vec_\$\$iadd_constant16_i: add a .....	vec_\$\$iadd_constant16_i
	vec_\$\$iadd_constant_i: add a .....	vec_\$\$iadd_constant_i
	vec_\$\$iadd_vector: add two .....	vec_\$\$iadd_vector
	vec_\$\$iadd_vector16: add two .....	vec_\$\$iadd_vector16
	vec_\$\$iadd_vector16_i: add .....	vec_\$\$iadd_vector16_i
	vec_\$\$iadd_vector_i: add .....	vec_\$\$iadd_vector_i
	vec_\$\$icopy: copy a 32-bit .....	vec_\$\$icopy
	vec_\$\$icopy16: copy a 16-bit .....	vec_\$\$icopy16
	vec_\$\$icopy16_i: copy a vector .....	vec_\$\$icopy16_i
	vec_\$\$icopy_i: copy a vector .....	vec_\$\$icopy_i
	vec_\$\$idot: return the dot .....	vec_\$\$idot
	vec_\$\$idot16: return the dot .....	vec_\$\$idot16
	vec_\$\$idot16_i: return the dot .....	vec_\$\$idot16_i
	vec_\$\$idot_i: return the dot .....	vec_\$\$idot_i
	vec_\$\$iinit: initialize a .....	vec_\$\$iinit
	vec_\$\$iinit16: initialize a .....	vec_\$\$iinit16
	vec_\$\$imat_mult: multiply two .....	vec_\$\$imat_mult
	vec_\$\$imat_mult16: multiply two .....	vec_\$\$imat_mult16
	vec_\$\$imat_multn: multiply two .....	vec_\$\$imat_multn
	vec_\$\$imat_multn16: multiply .....	vec_\$\$imat_multn16
	vec_\$\$imax: find the maximum .....	vec_\$\$imax
	vec_\$\$imax16: find the maximum .....	vec_\$\$imax16
	vec_\$\$imax16_i: find the .....	vec_\$\$imax16_i
	vec_\$\$imax_i: find the maximum .....	vec_\$\$imax_i
	vec_\$\$imult_add: scale and add .....	vec_\$\$imult_add

add one 16-bit integer vector/	vec_\$imult_add16: scale and .....	vec_\$imult_add16
add 16-bit vectors in/	vec_\$imult_add16_i: scale and .....	vec_\$imult_add16_i
add 32-bit vectors in/	vec_\$imult_add_i: scale and .....	vec_\$imult_add_i
a 32-bit integer vector by a/	vec_\$imult_constant: multiply .....	vec_\$imult_constant
multiply a 16-bit integer/	vec_\$imult_constant16: .....	vec_\$imult_constant16
multiply a vector in a 16-bit/	vec_\$imult_constant16_i: .....	vec_\$imult_constant16_i
multiply a vector in a 32-bit/	vec_\$imult_constant_i: .....	vec_\$imult_constant_i
single-precision vector	vec_\$init: initialize a .....	vec_\$init
	vec_\$intro: the Vector Library .....	vec_\$intro
32-bit integer vector by a/	vec_\$ipostmult: multiply a .....	vec_\$ipostmult
16-bit integer vector by a/	vec_\$ipostmult16: multiply a .....	vec_\$ipostmult16
32-bit integer vector by a/	vec_\$ipostmultn: multiply a .....	vec_\$ipostmultn
16-bit integer vector by a/	vec_\$ipostmultn16: multiply a .....	vec_\$ipostmultn16
32-bit vector by a 4x4/	vec_\$ipremult: multiply a .....	vec_\$ipremult
16-bit vector by a 4x4/	vec_\$ipremult16: multiply a .....	vec_\$ipremult16
32-bit integer vector by a/	vec_\$ipremultn: multiply a .....	vec_\$ipremultn
16-bit integer vector by a/	vec_\$ipremultn16: multiply a .....	vec_\$ipremultn16
integer vectors	vec_\$isub: subtract 32-bit .....	vec_\$isub
integer vectors	vec_\$isub16: subtract 16-bit .....	vec_\$isub16
integer vectors in matrixes	vec_\$isub16_i: subtract 16-bit .....	vec_\$isub16_i
integer vectors in matrixes	vec_\$isub_i: subtract 32-bit .....	vec_\$isub_i
a 32-bit integer vector	vec_\$isum: sum the elements of .....	vec_\$isum
of a 16-bit integer vector	vec_\$isum16: sum the elements .....	vec_\$isum16
elements of a vector in a/	vec_\$isum16_i: sum the .....	vec_\$isum16_i
of a vector in a 32-bit/	vec_\$isum_i: sum the elements .....	vec_\$isum_i
integer vectors	vec_\$iswap: swap two 32-bit .....	vec_\$iswap
integer vectors	vec_\$iswap16: swap two 16-bit .....	vec_\$iswap16
vectors in a 16-bit integer/	vec_\$iswap16_i: swap two .....	vec_\$iswap16_i
in a 32-bit integer matrix	vec_\$iswap_i: swap two vectors .....	vec_\$iswap_i
integer vector	vec_\$izero: zero a 32-bit .....	vec_\$izero
integer vector	vec_\$izero16: zero a 16-bit .....	vec_\$izero16
in a 16-bit integer matrix	vec_\$izero16_i: zero a vector .....	vec_\$izero16_i
a 32-bit integer matrix	vec_\$izero_i: zero a vector in .....	vec_\$izero_i
4x4 single-precision/	vec_\$imat_mult: multiply two .....	vec_\$imat_mult
single-precision matrixes	vec_\$imat_multn: multiply two .....	vec_\$imat_multn
absolute value in a/	vec_\$imax: find the maximum .....	vec_\$imax
absolute value in a vector/	vec_\$imax_i: find the maximum .....	vec_\$imax_i
one single-precision vector/	vec_\$imult_add: scale and add .....	vec_\$imult_add
single-precision vectors in/	vec_\$imult_add_i: scale and add .....	vec_\$imult_add_i
single-precision vector by a/	vec_\$imult_constant: multiply a .....	vec_\$imult_constant
a vector in a/	vec_\$imult_constant_i: multiply .....	vec_\$imult_constant_i
single-precision vector by a/	vec_\$ipostmult: multiply a .....	vec_\$ipostmult
single-precision vector by a/	vec_\$ipostmultn: multiply a .....	vec_\$ipostmultn
single-precision vector by a/	vec_\$ipremult: multiply a .....	vec_\$ipremult
single-precision vector by a/	vec_\$ipremultn: multiply a .....	vec_\$ipremultn
single-precision vector to a/	vec_\$isp_dp: copy a .....	vec_\$isp_dp
single-precision vector to a/	vec_\$isp_dp_i: copy a .....	vec_\$isp_dp_i
single-precision vectors	vec_\$isub: subtract .....	vec_\$isub
single-precision vectors in/	vec_\$isub_i: subtract .....	vec_\$isub_i
a single-precision vector	vec_\$isum: sum the elements of .....	vec_\$isum
of a vector in a/	vec_\$isum_i: sum the elements .....	vec_\$isum_i

single-precision vectors	vec_\$\$swap: swap two	vec_\$\$swap
in a single-precision matrix	vec_\$\$swap_i: swap two vectors	vec_\$\$swap_i
/multiply a double-precision	vector by a 4x4 matrix	vec_\$\$dpostmult
/multiply a double-precision	vector by a 4x4 matrix	vec_\$\$dpremult
/multiply a 32-bit integer	vector by a 4x4 matrix	vec_\$\$ipostmult
/multiply a 16-bit integer	vector by a 4x4 matrix	vec_\$\$ipostmult16
/multiply a 32-bit integer	vector by a 4x4 matrix	vec_\$\$ipremult
/multiply a 16-bit integer	vector by a 4x4 matrix	vec_\$\$ipremult16
/multiply a single-precision	vector by a 4x4 matrix	vec_\$\$postmult
/multiply a single-precision	vector by a 4x4 matrix	vec_\$\$spremult
/multiply a double-precision	vector by a matrix	vec_\$\$dpostmultn
/multiply a double-precision	vector by a matrix	vec_\$\$dpremultn
/multiply a 32-bit integer	vector by a matrix	vec_\$\$ipostmultn
/multiply a 16-bit integer	vector by a matrix	vec_\$\$ipostmultn16
/multiply a 32-bit integer	vector by a matrix	vec_\$\$ipremultn
/multiply a 16-bit integer	vector by a matrix	vec_\$\$ipremultn16
/multiply a single-precision	vector by a matrix	vec_\$\$spostmultn
/multiply a single-precision	vector by a matrix	vec_\$\$spremultn
/multiply a double-precision	vector by a scalar	vec_\$\$dmult_constant
/multiply a 32-bit integer	vector by a scalar	vec_\$\$simult_constant
/multiply a 16-bit integer	vector by a scalar	vec_\$\$simult_constant16
/multiply a single-precision	vector by a scalar	vec_\$\$smult_constant
/maximum absolute value in a	vector from a 16-bit integer/	vec_\$\$simax16_i
/maximum absolute value in a	vector from a 32-bit integer/	vec_\$\$simax_i
matrix/ vec_\$\$dp_sp_i: copy a	vector from a double-precision	vec_\$\$dp_sp_i
/maximum absolute value in a	vector from a double-precision/	vec_\$\$sdmax_i
/maximum absolute value in a	vector from a single-precision/	vec_\$\$smax_i
matrix/ vec_\$\$scopy16_i: copy a	vector from one 16-bit integer	vec_\$\$scopy16_i
matrix/ vec_\$\$scopy_i: copy a	vector from one 32-bit integer	vec_\$\$scopy_i
vec_\$\$scopy_i: copy a	vector from one/	vec_\$\$scopy_i
vec_\$\$scopy_i: copy a	vector from one/	vec_\$\$scopy_i
matrix by a scalar /multiply a	vector in a 16-bit integer	vec_\$\$simult_constant16_i
matrix /add a scalar to a	vector in a 16-bit integer	vec_\$\$siadd_constant16_i
matrix /sum the elements of a	vector in a 16-bit integer	vec_\$\$sisum16_i
matrix vec_\$\$zero16_i: zero a	vector in a 16-bit integer	vec_\$\$zero16_i
matrix by a scalar /multiply a	vector in a 32-bit integer	vec_\$\$simult_constant_i
matrix /add a scalar to a	vector in a 32-bit integer	vec_\$\$siadd_constant_i
matrix /sum the elements of a	vector in a 32-bit integer	vec_\$\$sisum_i
matrix vec_\$\$szero_i: zero a	vector in a 32-bit integer	vec_\$\$szero_i
matrix by a scalar /multiply a	vector in a double-precision	vec_\$\$sdmult_constant_i
matrix /add a scalar to a	vector in a double-precision	vec_\$\$sdadd_constant_i
matrix /sum the elements of a	vector in a double-precision	vec_\$\$sdsum_i
matrix vec_\$\$sdzero_i: zero a	vector in a double-precision	vec_\$\$sdzero_i
matrix by a scalar /multiply a	vector in a single-precision	vec_\$\$smult_constant_i
matrix /add a scalar to a	vector in a single-precision	vec_\$\$sadd_constant_i
matrix /sum the elements of a	vector in a single-precision	vec_\$\$ssum_i
matrix vec_\$\$szero_i: zero a	vector in a single-precision	vec_\$\$szero_i
/vector to a double-precision	vector in matrixes	vec_\$\$ssp_dp_i
vec_\$\$sintro: the	Vector Library	vec_\$\$sintro
get the program argument	vector pgm_\$\$get_args:	pgm_\$\$get_args
/copy a single-precision	vector to a double-precision/	vec_\$\$ssp_dp_i

/copy a single-precision	vector to a double-precision/	vec_\$sp_dp
/copy a double-precision	vector to a single-precision/	vec_\$dp_sp
and add one double-precision	vector to another /scale	vec_\$dmult_add
and add one 32-bit integer	vector to another /scale	vec_\$imult_add
and add one 16-bit integer	vector to another /scale	vec_\$imult_add16
and add one single-precision	vector to another /scale	vec_\$mult_add
a scalar to a single-precision	vector vec_\$add_constant: add	vec_\$add_constant
copy a single-precision	vector vec_\$copy:	vec_\$copy
a scalar to a double-precision	vector /add	vec_\$dadd_constant
copy a double-precision	vector vec_\$dcopy:	vec_\$dcopy
initialize a double-precision	vector vec_\$dinit:	vec_\$dinit
value in a double-precision	vector /the maximum absolute	vec_\$dmax
vector to a single-precision	vector /a double-precision	vec_\$dp_sp
elements of a double-precision	vector vec_\$dsum: sum the	vec_\$dsum
zero a double-precision	vector vec_\$dzero:	vec_\$dzero
constant to a 32-bit integer	vector /add a scalar	vec_\$iadd_constant
constant to a 16-bit integer	vector /add a scalar	vec_\$iadd_constant16
copy a 32-bit integer	vector vec_\$icopy:	vec_\$icopy
copy a 16-bit integer	vector vec_\$icopy16:	vec_\$icopy16
initialize a 32-bit integer	vector vec_\$iinit:	vec_\$iinit
initialize a 16-bit integer	vector vec_\$iinit16:	vec_\$iinit16
value in a 32-bit integer	vector /the maximum absolute	vec_\$imax
value in a 16-bit integer	vector /the maximum absolute	vec_\$imax16
initialize a single-precision	vector vec_\$init:	vec_\$init
elements of a 32-bit integer	vector vec_\$isum: sum the	vec_\$isum
elements of a 16-bit integer	vector vec_\$isum16: sum the	vec_\$isum16
zero a 32-bit integer	vector vec_\$izero:	vec_\$izero
zero a 16-bit integer	vector vec_\$izero16:	vec_\$izero16
value in a single-precision	vector /the maximum absolute	vec_\$imax
vector to a double-precision	vector /a single-precision	vec_\$sp_dp
elements of a single-precision	vector vec_\$sum: sum the	vec_\$sum
zero a single-precision	vector vec_\$zero:	vec_\$zero
/return the dot product of two	vectors in 16-bit integer/	vec_\$idot16_i
/return the dot product of two	vectors in 32-bit integer/	vec_\$idot_i
vec_\$iswap16_i: swap two	vectors in a 16-bit integer/	vec_\$iswap16_i
matrix vec_\$iswap_i: swap two	vectors in a 32-bit integer	vec_\$iswap_i
matrix vec_\$dswap_i: swap two	vectors in a double-precision	vec_\$dswap_i
matrix vec_\$swap_i: swap two	vectors in a single-precision	vec_\$swap_i
/return the dot product of two	vectors in double-precision/	vec_\$ddot_i
/scale and add double-precision	vectors in matrixes	vec_\$dmult_add_i
/subtract double-precision	vectors in matrixes	vec_\$dsub_i
/scale and add 16-bit	vectors in matrixes	vec_\$imult_add16_i
/scale and add 32-bit	vectors in matrixes	vec_\$imult_add_i
/subtract 16-bit integer	vectors in matrixes	vec_\$isub16_i
/subtract 32-bit integer	vectors in matrixes	vec_\$isub_i
/scale and add single-precision	vectors in matrixes	vec_\$mult_add_i
/subtract single-precision	vectors in matrixes	vec_\$sub_i
/return the dot product of two	vectors in single-precision/	vec_\$idot_i
vec_\$iadd_vector16_i: add	vectors in two 16-bit integer/	vec_\$iadd_vector16_i
vec_\$iadd_vector_i: add	vectors in two 32-bit integer/	vec_\$iadd_vector_i
vec_\$dadd_vector_i: add	vectors in two/	vec_\$dadd_vector_i

vec_ \$sadd_vector_i: add	vectors in two/	vec_ \$sadd_vector_i
add two single-precision	vectors vec_ \$sadd_vector:	vec_ \$sadd_vector
add two double-precision	vectors vec_ \$dadd_vector:	vec_ \$dadd_vector
of two double-precision	vectors /the dot product	vec_ \$ddot
of two single-precision	vectors /the dot product	vec_ \$dot
subtract double-precision	vectors vec_ \$dsub:	vec_ \$dsub
swap two double-precision	vectors vec_ \$dswap:	vec_ \$dswap
add two 32-bit integer	vectors vec_ \$iadd_vector:	vec_ \$iadd_vector
add two 16-bit integer	vectors vec_ \$iadd_vector16:	vec_ \$iadd_vector16
product of two 32-bit integer	vectors /return the dot	vec_ \$idot
product of two 16-bit integer	vectors /return the dot	vec_ \$idot16
subtract 32-bit integer	vectors vec_ \$isub:	vec_ \$isub
subtract 16-bit integer	vectors vec_ \$isub16:	vec_ \$isub16
swap two 32-bit integer	vectors vec_ \$iswap:	vec_ \$iswap
swap two 16-bit integer	vectors vec_ \$iswap16:	vec_ \$iswap16
subtract single-precision	vectors vec_ \$sub:	vec_ \$sub
swap two single-precision	vectors vec_ \$swap:	vec_ \$swap
single-precision vector	vec_ \$zero: zero a	vec_ \$zero
a single-precision matrix	vec_ \$zero_i: zero a vector in	vec_ \$zero_i
from a string	vfmt_ \$decode10: formatted read	vfmt_ \$decode10
from a string	vfmt_ \$decode2: formatted read	vfmt_ \$decode2
from a string	vfmt_ \$decode5: formatted read	vfmt_ \$decode5
to a string	vfmt_ \$encode: formatted write	vfmt_ \$encode
write to a string	vfmt_ \$encode10: formatted	vfmt_ \$encode10
to a string	vfmt_ \$encode2: formatted write	vfmt_ \$encode2
to a string	vfmt_ \$encode5: formatted write	vfmt_ \$encode5
formatting package	vfmt_ \$intro: variable	vfmt_ \$intro
from standard input	vfmt_ \$read10: formatted read	vfmt_ \$read10
from standard input	vfmt_ \$read2: formatted read	vfmt_ \$read2
from standard input	vfmt_ \$read5: formatted read	vfmt_ \$read5
from a stream	vfmt_ \$rs10: formatted read	vfmt_ \$rs10
a stream	vfmt_ \$rs2: formatted read from	vfmt_ \$rs2
a stream	vfmt_ \$rs5: formatted read from	vfmt_ \$rs5
to standard output	vfmt_ \$write: formatted write	vfmt_ \$write
to standard output	vfmt_ \$write10: formatted write	vfmt_ \$write10
to standard output	vfmt_ \$write2: formatted write	vfmt_ \$write2
to standard output	vfmt_ \$write5: formatted write	vfmt_ \$write5
stream	vfmt_ \$ws: formatted write to a	vfmt_ \$ws
a stream	vfmt_ \$ws10: formatted write to	vfmt_ \$ws10
a stream	vfmt_ \$ws2: formatted write to	vfmt_ \$ws2
a stream	vfmt_ \$ws5: formatted write to	vfmt_ \$ws5
permanently map an object	via its XOID ms_ \$mapl_stream:	ms_ \$mapl_stream
a new pad and a window to	view it /create	pad_ \$create_window
/make a window	visible and full-size	pad_ \$select_window
mapped objects are on the same	volume /find out if two	ms_ \$neighbors
during I/O ec2_ \$wait_slow_io:	wait for an eventcount trigger	ec2_ \$wait_slow_io
ec2_ \$wait:	wait for an eventcount trigger	ec2_ \$wait
or/ ec2_ \$wait_svc:	wait for an eventcount trigger	ec2_ \$wait_svc
time_ \$wait:	wait for an interval	time_ \$wait
ipc_ \$wait:	wait for an IPC datagram	ipc_ \$wait
pgm_ \$proc_wait:	wait on a process	pgm_ \$proc_wait

closes pad_\$edit_wait:	wait until an edit window .....	pad_\$edit_wait
moves pad_\$icon_wait:	wait until an icon expands or .....	pad_\$icon_wait
/read a channel without	waiting .....	mbx_\$cond_get_rec_chan
/read a set of channels without	waiting .....	mbx_\$cond_get_rec_chan_set
read a message without	waiting mbx_\$get_conditional: .....	mbx_\$get_conditional
send a partial message without	waiting mbx_\$put_chr_cond: .....	mbx_\$put_chr_cond
send a message without	waiting mbx_\$put_rec_cond: .....	mbx_\$put_rec_cond
cal_\$weekday: return the	weekday for a date .....	cal_\$weekday
/add or remove a	window border .....	pad_\$set_border
wait until an edit	window closes pad_\$edit_wait: .....	pad_\$edit_wait
a new pad and associated	window icon /create .....	pad_\$create_icon
pad_\$make_icon: change a	window into an icon .....	pad_\$make_icon
pad_\$make_invisible: make a	window invisible .....	pad_\$make_invisible
/find out whether a	window is an icon .....	pad_\$is_icon
display region occupied by a	window /get the total .....	pad_\$inq_full_window
out the pad region under the	window pad_\$inq_view: find .....	pad_\$inq_view
pop or push a	window pad_\$pop_push_window: .....	pad_\$pop_push_window
define the pad region under a	window pad_\$set_view: .....	pad_\$set_view
create a new pad and	window pane pad_\$create: .....	pad_\$create
/establish a	window region .....	pad_\$set_full_window
/create a new pad and a	window to view it .....	pad_\$create_window
pad_\$select_window: make a	window visible and full-size .....	pad_\$select_window
pad_\$set_auto_close: close a	window when its pad closes .....	pad_\$set_auto_close
pad_\$inq_windows: list the	windows on a pad .....	pad_\$inq_windows
name_\$get_wdir_lc: get the	working directory .....	name_\$get_wdir_lc
name_\$set_wdir: sets the	working directory .....	name_\$set_wdir
/get the current	working or naming directory .....	ios_\$get_dir_lc
/change the current	working or naming directory .....	ios_\$set_dir
vfmt_\$ws: formatted	write to a stream .....	vfmt_\$ws
vfmt_\$ws10: formatted	write to a stream .....	vfmt_\$ws10
vfmt_\$ws2: formatted	write to a stream .....	vfmt_\$ws2
vfmt_\$ws5: formatted	write to a stream .....	vfmt_\$ws5
vfmt_\$encode: formatted	write to a string .....	vfmt_\$encode
vfmt_\$encode10: formatted	write to a string .....	vfmt_\$encode10
vfmt_\$encode2: formatted	write to a string .....	vfmt_\$encode2
vfmt_\$encode5: formatted	write to a string .....	vfmt_\$encode5
ios_\$put:	write to an object .....	ios_\$put
vfmt_\$write: formatted	write to standard output .....	vfmt_\$write
vfmt_\$write10: formatted	write to standard output .....	vfmt_\$write10
vfmt_\$write2: formatted	write to standard output .....	vfmt_\$write2
vfmt_\$write5: formatted	write to standard output .....	vfmt_\$write5
map an object via its	XOID /permanently .....	ms_\$mapl_stream
task_\$yield:	yield the processor .....	task_\$yield
vec_\$izero16:	zero a 16-bit integer vector .....	vec_\$izero16
vec_\$izero:	zero a 32-bit integer vector .....	vec_\$izero
vec_\$dzero:	zero a double-precision vector .....	vec_\$dzero
vec_\$zero:	zero a single-precision vector .....	vec_\$zero
integer/ vec_\$izero16_i:	zero a vector in a 16-bit .....	vec_\$izero16_i
integer matrix vec_\$izero_i:	zero a vector in a 32-bit .....	vec_\$izero_i
vec_\$dzero_i:	zero a vector in a/ .....	vec_\$dzero_i
single-precision/ vec_\$zero_i:	zero a vector in a .....	vec_\$zero_i

decode an ASCII time zone cal\_\$decode\_ascii\_tzdiff: ..... cal\_\$decode\_ascii\_tzdiff  
get the local time zone cal\_\$get\_info: ..... cal\_\$get\_info  
set the local time zone cal\_\$write\_timezone: ..... cal\_\$write\_timezone

---

**Contents**

intro .....	PFM-1
pfm_\$cleanup .....	PFM-4
pfm_\$enable .....	PFM-6
pfm_\$error_trap .....	PFM-7
pfm_\$establish_fault_handler .....	PFM-8
pfm_\$inhibit .....	PFM-10
pfm_\$release_fault_handler .....	PFM-11
pfm_\$reset_cleanup .....	PFM-12
pfm_\$rls_cleanup .....	PFM-13
pfm_\$signal .....	PFM-14



**NAME**

intro – managing faults

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/pfm.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pfm.ins.ftn'
```

**DESCRIPTION**

The **pfm\_\$** calls allow programs to manage signals, faults, and exceptions. Programs can manage faults by establishing clean-up handlers and fault handlers.

**Clean-up Handlers**

A clean-up handler is a piece of code that ensures a program terminates gracefully when it receives a fatal error. A clean-up handler begins with a **pfm\_\$cleanup** call, and usually ends with a call to **pfm\_\$signal** or **pgm\_\$exit**, though it can also simply continue back into the program after the clean-up code.

A clean-up handler is not entered until all fault handlers established for a fault have returned. If there is more than one established clean-up handler for a program, the most recently established clean-up handler is entered first, followed by the next most recently established clean-up handler, and so on to the first established clean-up handler if necessary.

There is a default clean-up handler invoked after all user-defined handlers have completed. It releases any resources still held by the program, before returning control to the process that invoked it.

**Fault Handlers**

A fault handler is a function supplied by the user that is called when a fault occurs. When a fault occurs that has a handler established for it, the system interrupts the faulted program and calls the designated fault handler. When the fault handler returns, processing resumes where the program was interrupted.

There are three types of fault handlers. By default a handler is called only when a signal is received at the program level at which it was established. A "multilevel" fault handler is called whenever a signal is received at its or any subordinate program level. A "backstop" fault handler is called only after all other non-backstop handlers have returned.

All fault handlers must take one argument of type **pfm\_\$fault\_rec\_t** and return a value of type **pfm\_\$fn\_func\_val\_t**, but their implementation is otherwise left to the user. If there is more than one established handler for a fault, the most recent nonbackstop handler is called first, followed by the next most recent, if necessary, and so on until all

the applicable non-backstop handlers have returned. Then any backstop handlers are called, if necessary, in the same last-established/first-called order.

#### Constants

##### **pfm\_\$all\_faults**

Passed to **pfm\_\$establish\_fault\_handler** to establish a handler for all possible faults.

#### Data Types

##### **pfm\_\$cleanup\_rec**

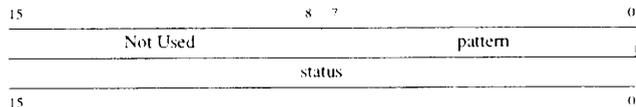
A record type for passing process context among clean-up handler calls. It is an opaque data type.

##### **pfm\_\$fault\_func\_p\_t**

A pointer to a fault handler function.

##### **pfm\_\$fault\_rec\_t**

A record type for passing status information among fault handlers. Only the **status** field is accessible to fault handler functions. The following diagram illustrates the data type:



**pattern** Reserved for system use.

**status** A fault status of type **status\_\$t**.

##### **pfm\_\$fh\_func\_val\_t**

A value returned by a fault handler function. It specifies the action to be taken when the handler completes. It can be one of the following values:

##### **pfm\_\$continue\_fault\_handling**

Pass the fault to the next handler.

##### **pfm\_\$return\_to\_faulting\_code**

Return control to the faulted program.

##### **pfm\_\$fh\_handle\_t**

A pointer to a fault handler function.

##### **pfm\_\$fh\_opt\_set\_t**

A small set of options when establishing a fault handler. It can be any combination of the following values:

**pfm\_\$fh\_multi\_level**

Establish a "multilevel" handler for faults encountered at the current and all subordinate program levels.

**pfm\_\$fh\_backstop**

Establish a "backstop" handler that is called after all nonbackstop handlers have been called.

By default, fault handlers are called consecutively on a last-in/first-out basis, starting with the most recently established handler and ending with the first fault handler established in the current program level. To establish a default handler in Pascal, pass an empty set [] for *fh\_options*; in C, pass 0.

**Errors****pfm\_\$bad\_rls\_order**

Attempted to release a clean-up handler out of order.

**pfm\_\$cleanup\_not\_found**

There is no pending clean-up handler.

**pfm\_\$cleanup\_set**

A clean-up handler was established successfully.

**pfm\_\$cleanup\_set\_signalled**

Attempted to use **pfm\_\$cleanup\_set** as a signal.

**pfm\_\$fh\_not\_found**

Attempted to release non-existent fault handler.

**pfm\_\$fh\_wrong\_level**

Attempted to release fault handler at wrong level.

**pfm\_\$invalid\_cleanup\_rec**

Passed an invalid clean-up record to a call.

**pfm\_\$no\_space**

Cannot allocate storage for a clean-up handler.

**SEE ALSO**

fault\_\$intro.

## NAME

**pfm\_\$cleanup** – establish a clean-up handler

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>
```

```
status $t pfm_$cleanup(pfm_$cleanup_rec *cleanup_record)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';
```

```
function pfm_$cleanup(
    out cleanup_record: pfm_$cleanup_rec; status $t;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pfm.ins.ftn'
```

```
integer*4 status, cleanup_record(16)
```

```
status = pfm_$cleanup(cleanup_record)
```

## DESCRIPTION

**Pfm\_\$cleanup** establishes a clean-up handler that is executed when a fault occurs. A clean-up handler is a piece of code executed before a program exits when a signal is received by the process. The clean-up handler begins where **pfm\_\$cleanup** is called; the **pfm\_\$cleanup** call registers an entry point with the system where program execution resumes when a fault occurs. When a fault occurs, execution resumes after the most recent call to **pfm\_\$cleanup**.

There can be more than one clean-up handler in a program. Multiple clean-up handlers are executed consecutively on a last-in/first-out basis, starting with the most recently established handler and ending with the first clean-up handler. The system provides a default clean-up handler established at program invocation. The default clean-up handler is always called last, just before a program exits, and releases any system resources still held, before returning control to the process that invoked the program.

When called to establish a clean-up handler, **pfm\_\$cleanup** returns the status **pfm\_\$cleanup\_set** to indicate the clean-up handler was successfully established. When the clean-up handler is entered in response to a fault signal, **pfm\_\$cleanup** effectively returns the value of the fault that triggered the handler.

*cleanup\_record*

A record of the context when **pfm\_\$cleanup** is called. A program should treat this as an opaque data structure, and not try to alter or copy its contents. It is needed by **pfm\_\$rls\_cleanup** and **pfm\_\$reset\_cleanup** to restore the context of the calling process at the clean-up handler entry point.

## NOTES

Clean-up handler code runs with asynchronous faults inhibited. When `pfm_$cleanup` returns something other than `pfm_$cleanup_set`, indicating that a fault has occurred, there are four possible ways to leave the clean-up code:

- The program can call `pfm_$signal` to start the next clean-up handler with a different fault signal.
- The program can call `pgm_$exit` to start the next clean-up handler with the same fault signal.
- The program can continue with the code following the clean-up handler. It should generally call `pfm_$enable` to re-enable asynchronous faults. Execution continues from the end of the clean-up handler code; it does not resume where the fault signal was received.
- The program can re-establish the handler by calling `pfm_$reset_cleanup` before proceeding.

## SEE ALSO

`fault_$intro`, `pfm_$error_trap`, `pfm_$signal`.

## NAME

**pfm\_\$enable** – enable asynchronous faults

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>
```

```
void pfm_$enable(void)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';
```

```
procedure pfm_$enable;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pfm.ins.ftn'
```

```
call pfm_$enable
```

## DESCRIPTION

**Pfm\_\$enable** enables asynchronous faults after they have been inhibited by a call to **pfm\_\$inhibit**. **Pfm\_\$enable** causes the operating system to pass asynchronous faults on to the calling process.

While faults are inhibited, the operating system holds at most one asynchronous fault. Consequently, when **pfm\_\$enable** returns, there can be at most one fault waiting on the process. If more than one fault was received between calls to **pfm\_\$disable** and **pfm\_\$enable**, the process receives the first asynchronous fault received while faults were inhibited.

## NAME

**pfm\_error\_trap** – simulate a fault and save a traceback

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>
```

```
void pfm_error_trap(status_t &fault_signal)
```

## SYNOPSIS (Pascal)

```
%include 'sys/ins/base.ins.pas';
%include 'sys/ins/pfm.ins.pas';
```

```
procedure pfm_error_trap(in fault_signal: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include 'sys/ins/base.ins.ftn'
%include 'sys/ins/pfm.ins.ftn'
```

```
integer*4 fault_signal
```

```
call pfm_error_trap(fault_signal)
```

## DESCRIPTION

**Pfm\_error\_trap** signals the fault specified by *fault\_signal* to the calling process, and saves a traceback of the calling sequence. It is especially useful when debugging a program, or in writing debuggers and monitoring utilities.

*fault\_signal*

A fault code.

## NOTES

**Pfm\_error\_trap** does not return when successful.

## SEE ALSO

fault\_intro, pfm\_signal.

## NAME

**pfm\_\$establish\_fault\_handler** – establish a fault handler

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>
```

```
pfm_$fh_handle_t pfm_$establish_fault_handler(
    int &target_status,
    pfm_$fh_opt_set_t &fh_options,
    pfm_$fault_func_p_t &function_ptr,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';
```

```
function pfm_$establish_fault_handler(
    in target_status: integer32;
    in fh_options: pfm_$fh_opt_set_t;
    in function_ptr: pfm_$fault_func_p_t;
    out status: status_t): pfm_$fh_handle_t;
```

## DESCRIPTION

**pfm\_\$establish\_fault\_handler** establishes the function at *function\_ptr* as a handler for the faults specified by *target\_status*, and returns a handle for it.

*target\_status*

A value specifying the type of fault the function at *function\_ptr* should handle. Set *target\_status* to **pfm\_\$all\_faults** to establish a handler for all possible faults. To establish a handler for all faults produced by a set of Domain/OS calls, use one of that manager's predefined completion status values with the **code** field set to 0.

*fh\_options*

This is a small set of options that limit the scope of the fault handler. Specify any combination of the following values:

**pfm\_\$fh\_multi\_level**

Establish a "multilevel" handler for faults encountered at the current and all subordinate program levels.

**pfm\_\$fh\_backstop**

Establish a "backstop" handler that is called after all nonbackstop handlers have been called.

By default, fault handlers are called consecutively on a last-in/first-out basis, starting with the most recently established handler and ending with the first fault handler established in the current program level. To

PFM\_\$ESTABLISH\_FAULT\_HANDLER

PFM\_\$ESTABLISH\_FAULT\_HANDLER

establish a default handler in Pascal, pass an empty set [] for *fh\_options*; in C, pass 0.

*function\_ptr*

The address of a function to establish as a fault handler.

*status* The completion status.

**NOTES**

A fault handler remains in effect until released by passing the returned handle to **pfm\_\$release\_fault\_handler**.

## NAME

**pfm\_\$(inhibit** – inhibit asynchronous faults

## SYNOPSIS (C)

```
#include <apollo/base.h>
```

```
#include <apollo/pfm.h>
```

```
void pfm_$(inhibit(void);
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
```

```
%include '/sys/ins/pfm.ins.pas';
```

```
procedure pfm_$(inhibit;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
%include '/sys/ins/pfm.ins.ftn'
```

```
call pfm_$(inhibit
```

## DESCRIPTION

**Pfm\_\$(inhibit** prevents asynchronous faults from being passed to the calling process. While faults are inhibited, the operating system holds at most one asynchronous fault. Consequently, a call to **pfm\_\$(inhibit** can result in the loss of some signals. For that and other reasons, it is good practice to inhibit faults only when absolutely necessary.

## NOTES

**Pfm\_\$(inhibit** has no effect on the processing of synchronous faults such as floating-point and overflow exceptions, access violations, and so on.

## SEE ALSO

**pfm\_\$(enable**.

## NAME

**pfm\_\$release\_fault\_handler** – release a fault handler

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>

void pfm_$release_fault_handler(
    pfm_$fh_handle_t &fh_handle,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';

procedure pfm_$release_fault_handler(
    in fh_handle: pfm_$fh_handle_t;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pfm.ins.ftn'

integer*4 fh_handle, status

call pfm_$release_fault_handler(fh_handle, status)
```

## DESCRIPTION

**Pfm\_\$release\_fault\_handler** releases the fault handler on *fh\_handle*. After **pfm\_\$release\_fault\_handler** returns, the function acting as the fault handler will no longer be called when the process receives the fault signal it was established to handle.

*fh\_handle*

A handle on the fault handler to be released.

*status* The completion status.

## NOTES

Releasing a fault handler established inside another fault handler may cause unpredictable results.

## NAME

**pfm\_\$reset\_cleanup** – reset a clean-up handler

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>

void pfm_$reset_cleanup(
    pfm_$cleanup_rec &cleanup_record,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';

procedure pfm_$reset_cleanup(
    in cleanup_record: pfm_$cleanup_rec;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pfm.ins.ftn'

integer*4 cleanup_record(16), status

call pfm_$reset_cleanup(cleanup_record, status)
```

## DESCRIPTION

**Pfm\_\$reset\_cleanup** re-establishes the clean-up handler last entered so that any subsequent errors enter it first. This procedure should only be used within clean-up handler code.

*cleanup\_record*

A record of the context at the clean-up handler entry point. It is supplied by **pfm\_\$cleanup**, when the clean-up handler is first established.

*status* The completion status.

## NAME

**pfm\_srls\_cleanup** – release clean-up handlers stack.

## SYNOPSIS (C)

```
#include <apollo/base.h>
```

```
#include <apollo/pfm.h>
```

```
void pfm_srls_cleanup(
    pfm_cleanup_rec &cleanup_record,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
```

```
%include '/sys/ins/pfm.ins.pas';
```

```
procedure pfm_srls_cleanup(
    in cleanup_record: pfm_cleanup_rec;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
%include '/sys/ins/pfm.ins.ftn'
```

```
integer*4 cleanup_record(16), status
```

```
call pfm_srls_cleanup(cleanup_record, status)
```

## DESCRIPTION

**pfm\_srls\_cleanup** releases the clean-up handler associated with *cleanup\_record*, and all clean-up handlers established after it.

*cleanup\_record*

The clean-up record for the first clean-up handler to release.

*status*

The completion status. If *status* is **pfm\_bad\_rls\_order**, it means that the caller attempted to release a clean-up handler before releasing all handlers established after it. This status is only a warning; the intended clean-up handler is released, along with all clean-up handlers established after it.

## NAME

`pfm_$$ignal` – signal the calling process

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pfm.h>
```

```
void pfm_$$ignal(status_$$t &fault_$$ignal)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pfm.ins.pas';
```

```
procedure pfm_$$ignal(in fault_$$ignal: status_$$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pfm.ins.ftn'
```

```
integer*4 fault_$$ignal
```

```
call pfm_$$ignal(fault_$$ignal)
```

## DESCRIPTION

`Pfm_$$ignal` signals the fault specified by `fault_$$ignal` to the calling process. It is usually called to leave clean-up handlers.

`fault_$$ignal`

A fault code.

## NOTES

`Pfm_$$ignal` does not return when successful.

## SEE ALSO

`pfm_$$error_trap`.

---

Contents

intro .....	PGM-1
pgm_\$del_arg .....	PGM-4
pgm_\$exit .....	PGM-5
pgm_\$get_arg .....	PGM-6
pgm_\$get_args .....	PGM-8
pgm_\$get_ec .....	PGM-9
pgm_\$get_puid .....	PGM-11
pgm_\$invoke .....	PGM-12
pgm_\$make_orphan .....	PGM-15
pgm_\$proc_wait .....	PGM-16
pgm_\$set_severity .....	PGM-17



**NAME**

intro – the program manager

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

**DESCRIPTION**

The `pgm_$` calls allow programs to invoke other programs, pass system resources to child processes, and monitor their status.

**Constants****pgm\_\$error**

The severity level to indicate that an error occurred in an invoked program.

**pgm\_\$false**

A severity level to indicate that a condition was false.

**pgm\_\$internal\_fatal**

A severity level to indicate that an internal fatal error occurred in an invoked program.

**pgm\_\$max\_severity**

The highest exit severity level that can be returned by an invoked program.

**pgm\_\$ok**

A severity level to indicate that an invoked program completed successfully.

**pgm\_\$output\_invalid**

A severity level to indicate that the output from an invoked program is invalid.

**pgm\_\$program\_faulted**

A severity level to indicate that an invoked program was faulted.

**pgm\_\$true**

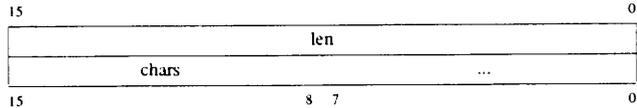
A severity level to indicate that a condition was true.

**pgm\_\$warning**

A severity level to indicate a warning.

**Data Types**

**pgm\_\$arg** A record for passing program arguments. The following diagram illustrates the `pgm_$arg` data type:



**len** The number of significant bytes in **chars**.

**chars** A character array to hold the text of the argument.

**pgm\_\$argv**

An array of program arguments.

**pgm\_\$argv\_ptr**

A pointer to a program argument.

**pgm\_\$connv**

An array for passing stream IDs to invoked programs.

**pgm\_\$ec\_key**

A key specifying a process eventcount. Currently, **pgm\_\$child\_proc** is the only valid value. It designates an eventcount that advances when a child process terminates.

**pgm\_\$mode**

A small set type to specify the mode in which to invoke a program. It can take on any combination of the following predefined values:

**pgm\_\$back\_ground**

The invoked program executes independently of the invoking process, and there is no valid process handle for it. Consequently, there is no mechanism by which the new process can communicate its exit status to the invoking process. Use this mode when the exit status of the invoked process is not important.

**pgm\_\$extra\_proc**

This mode forces the invoked program to run in a separate process.

**pgm\_\$obj\_only**

In this mode, the invoke call will fail if the invoked program is not a pre-SR10 object; that is, an object of type **obj**.

**pgm\_\$wait**

The invoking program is suspended while the invoked program executes. The termination status of the invoked program is supplied in the completion status of **pgm\_\$invoke** when it returns.

**pgm\_\$proc**

A pointer type used as a process handle.

**Errors****pgm\_\$no\_arg**

There is no argument in the argument vector at the specified position.

**pgm\_\$arg\_too\_big**

The argument requested is too big to fit in the buffer allocated for it.

**pgm\_\$bad\_connv**

Attempted to pass too many stream IDs in the connection vector.

**pgm\_\$no\_entry\_point**

There is no entry point in the invoked program.

**pgm\_\$not\_a\_program**

Attempted to invoke a file that is not an object.

**pgm\_\$on\_signal\_stack**

Attempted to invoke a program inprocess while running on the signal stack. See BSD sigstack(2).

**pgm\_\$option\_conflict**

Specified incompatible options to **pgm\_\$invoke**.

**pgm\_\$process\_vforked**

Cannot call **pgm\_\$invoke** from a process created by a BSD vfork(2).

**pgm\_\$wrong\_format**

Attempted to invoke a COFF object module with the **pgm\_\$obj\_only** option.

## NAME

**pgm\_\$del\_arg** – delete a program argument

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$del_arg(short &arg_number)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$del_arg(in arg_number: integer);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*2 arg_number
```

```
call pgm_$del_arg(arg_number)
```

## DESCRIPTION

**Pgm\_\$del\_arg** deletes an argument from the argument vector.

*arg\_number*

The index into the argument vector of the argument to delete. Arguments are numbered starting from 0, where argument 0 is usually the program invocation name. When **pgm\_\$del\_arg** returns, arguments coming after position *arg\_number* in the argument vector will be shifted down in position.

**NAME**

**pgm\_\$exit** – exit a program

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$exit(void)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$exit;
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
call pgm_$exit
```

**DESCRIPTION**

**Pgm\_\$exit** exits from the calling program and returns control to the process that invoked it. When **pgm\_\$exit** is called any files left open by the program are closed, any storage acquired is released, and asynchronous faults are re-enabled if they were inhibited by the calling program.

**NOTES**

**Pgm\_\$exit** calls **pfm\_\$signal** with a fault code equal to the last severity level set by **pgm\_\$set\_severity**, or **pgm\_\$ok** (the default) if **pgm\_\$set\_severity** was not called.

## NAME

`pgm_$get_arg` – get a program argument

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
short pgm_$get_arg(
    short &arg_number,
    char *arg_buffer,
    status $t *status,
    short &buffer_length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
function pgm_$get_arg(
    in arg_number: integer;
    out arg_buffer: univ pgm_$name;
    out status: status $t;
    in buffer_length: integer): integer;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*4 status
integer*2 arg_length, arg_number, buffer_length
character arg_buffer*128
```

```
arg_length = pgm_$get_arg(arg_number, arg_buffer, status, buffer_length)
```

## DESCRIPTION

`Pgm_$get_arg` writes an argument from the caller's argument vector into the buffer at `arg_buffer` and returns its length.

*arg\_number*

The number of the argument to get.

*arg\_buffer*

A buffer allocated to receive the argument.

*status*

The completion status. If *status* is `pgm_$arg_too_big`, the argument was more than *buffer\_length* bytes long and was truncated to fit.

*buffer\_length*

The number of bytes allocated to receive the argument. `Pgm_$get_arg` will not write more than *buffer\_length* bytes into *arg\_buffer*. If the value of

PGM\_\$GET\_ARG

PGM\_\$GET\_ARG

*buffer\_length* is less than the returned argument length, the argument was truncated to fit in *arg\_buffer*.

## NAME

`pgm_$get_args` – get the program argument vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$get_args(
    short *argument_count,
    pgm_$argv_ptr *arg_vector_ptr)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$get_args(
    out argument_count: integer;
    out arg_vector_ptr: pgm_$argv_ptr);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*2 argument_count, arg_len
integer*4 arg_ptr_vector(128)
character arg_chars*128
```

```
integer*4 arg_ptr, arg_vector_ptr
pointer /arg_ptr/ arg_len, arg_chars
pointer /arg_vector_ptr/ arg_ptr_vector
```

```
call pgm_$get_args(argument_count, arg_vector_ptr)
```

## DESCRIPTION

`Pgm_$get_args` supplies a pointer to the caller's argument vector and the number of arguments in it. The argument vector is an array of type `pgm_$arg`.

*argument\_count*

The number of arguments in the argument vector.

*arg\_vector\_ptr*

A pointer to the argument vector.

## NAME

`pgm_$get_ec` – get a process eventcount

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>

void pgm_$get_ec(
    pgm_$proc &process_handle,
    pgm_$ec_key &ec_key,
    ec2_$ptr_t *ec_pointer,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';

procedure pgm_$get_ec(
    in process_handle: pgm_$proc;
    in ec_key: pgm_$ec_key;
    out ec_pointer: ec2_$ptr_t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'

integer*4 status, process_handle, ec_value
integer*2 ec_key, event(3)

equivalence (ec_value, event(1))

integer*4 ec_pointer
pointer /ec_pointer/ event

call pgm_$get_ec(process_handle, ec_key, ec_pointer, status)
```

## DESCRIPTION

`Pgm_$get_ec` supplies a pointer to an eventcount that advances when the process on `process_handle` terminates. When a child process is created, the eventcount value is 0. When a child process terminates, the process eventcount value is 1.

*process\_handle*

A process handle of a child process. A process handle is not valid for programs invoked in background mode.

*ec\_key* An enumerated value specifying which process eventcount `pgm_$get_ec` should supply. The only allowable value currently is `pgm_$child_proc`.

PGM\_\$GET\_EC

PGM\_\$GET\_EC

*ec\_pointer*

An pointer to the eventcount.

*status* The completion status.

## NAME

`pgm_$get_puid` – get a process UID

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$get_puid(
    pgm_$proc &process_handle,
    uid_$t *process_uid,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$get_puid(
    in process_handle: pgm_$proc;
    out process_uid: uid_$t;
    out status: status_$t)
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*4 process_handle, process_uid(2), status
```

```
call pgm_$get_puid(process_handle, process_uid, status)
```

## DESCRIPTION

`Pgm_$get_puid` supplies the process UID (Unique Identifier) of the process on *process\_handle*.

*process\_handle*

A process handle for a child process. A process handle is not valid if the program was invoked in background mode.

*process\_uid*

The process UID of the process on *process\_handle*.

*status*

The completion status.

## NAME

pgm\_\$invoke – invoke a program

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>

void pgm_$invoke(
    char *path_name,
    short &path_length,
    short &argument_count,
    pgm_$arg_ptr *argument_vector,
    short &stream_count,
    stream_$id_t *stream_vector,
    pgm_$mode &invoke_mode,
    pgm_$proc *process_handle,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$invoke(
    in path_name: univ name_$pname_t;
    in path_length: integer;
    in argument_count: integer;
    in argument_vector: univ pgm_$argv;
    in stream_count: integer;
    in stream_vector: univ pgm_$connv;
    in invoke_mode: pgm_$mode;
    out process_handle: univ pgm_$proc;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*4 process_handle, status
integer*2 path_length, argument_count, arg_len, stream_count
integer*2 stream_vector(128), invoke_mode
character path_name*1023, arg_chars*128
```

```
integer*4 arg_ptr, argument_vector(128)
pointer /arg_ptr/ arg_len, arg_chars
```

```
call pgm_$invoke(path_name, path_length, argument_count,
```

& *argument\_vector, stream\_count,*  
 & *stream\_vector, invoke\_mode,*  
 & *process\_handle, status)*

## DESCRIPTION

**Pgm\_\$invoke** invokes the program at *path\_name* and supplies a process handle for it in *process\_handle*. The behavior of an invoked program depends on the mode in which the program is invoked.

*path\_name*

The absolute pathname of the program to invoke.

*path\_length*

The number of bytes in *path\_name*.

*argument\_count*

The number of arguments in *argument\_vector*.

*argument\_vector*

An array of program arguments of type **pgm\_\$arg**.

*stream\_count*

The number of stream IDs in *stream\_vector*.

*stream\_vector*

An array of stream IDs to pass to the invoked program.

*invoke\_mode*

A small set specifying the mode in which to invoke the program. Choose a combination of the following values:

**pgm\_\$back\_ground**

The invoked program executes independently of the invoking process, and there is no valid process handle for it. Consequently, there is no mechanism by which the new process can communicate its exit status to the invoking process. Use this mode when the exit status of the invoked process is not important.

**pgm\_\$extra\_proc**

This mode forces the invoked program to run in a separate process.

**pgm\_\$obj\_only**

In this mode, the invoke call will fail if the invoked program is not a pre-SR10 object; that is, an object of type **obj**.

**pgm\_\$wait**

The invoked program is suspended while the invoked program executes. The termination status of the invoked program is supplied in the completion status of **pgm\_\$invoke** when it returns.

To invoke in default mode, specify 0 in C and FORTRAN, or [] in Pascal. When a program is invoked in default mode, **pgm\_\$invoke** supplies a process handle for the child process by which the parent process can monitor its status.

*process\_handle*

A process handle for the invoked program. The process handle is not valid if the program was invoked in background mode.

*status* The completion status. If **pgm\_\$invoke** was called to invoke a program in wait mode, then *status* is the completion status of the invoked program or one of the following predefined severity levels:

**pgm\_\$true**

The value of a tested condition is true.

**pgm\_\$false**

The value of a tested condition is false.

**pgm\_\$warning**

An unusual, but not fatal, condition was detected.

**pgm\_\$error**

There were syntactic or semantic errors in the input, but the output is structurally sound.

**pgm\_\$invalid\_output**

There were syntactic or semantic errors in the input, and the output is not structurally sound.

**pgm\_\$internal\_fatal**

A fatal internal error detected.

## NOTES

When a program is invoked in default mode, some of the parent's resources are reserved to keep track of the child processes whose handles are held by the parent process. A parent process that indiscriminately invokes programs in default mode will eventually exhaust system resources unless it orphans its child processes by calling **pgm\_\$make\_orphan**.

## SEE ALSO

**pm\_\$intro**.

## NAME

**pgm\_\$make\_orphan** – orphan a process

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$make_orphan(
    pgm_$proc &process_handle,
    uid_$t *process_uid,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$make_orphan(
    in process_handle: pgm_$proc;
    out process_uid: uid_$t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*4 process_handle, process_uid(2), status
```

```
call pgm_$make_orphan(process_handle, process_uid, status)
```

## DESCRIPTION

**Pgm\_\$make\_orphan** changes the process on *process\_handle* from a child into an orphan process. An orphan process executes in background mode, and runs independently of its parent process. A parent process can use **pgm\_\$make\_orphan** to change a child process invoked in default mode into a process running in background mode. When **pgm\_\$make\_orphan** returns, *process\_handle* is no longer valid.

*process\_handle*

The process handle of a child process.

*process\_uid*

The process UID of the orphaned process.

*status* The completion status.

## NAME

`pgm_$proc_wait` – wait on a process

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$proc_wait(
    pgm_$proc &process_handle,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$proc_wait(
    in process_handle: pgm_$proc;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*4 process_handle, status
```

```
call pgm_$proc_wait(process_handle, status)
```

## DESCRIPTION

`Pgm_$proc_wait` suspends the calling process until the process on *process\_handle* terminates. `Pgm_$proc_wait` returns with the completion status of the process waited for in *status*.

Using `pgm_invoke` to invoke a program in default mode and then calling `pgm_$proc_wait` is effectively the same as invoking a program in wait mode.

*process\_handle*

The process handle of the process to wait for. A process cannot wait on a program invoked in background mode.

*status* The child process completion status.

## NAME

**pgm\_\$set\_severity** – set the exit severity level

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pgm.h>
```

```
void pgm_$set_severity(pgm_$mode &severity_level)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pgm.ins.pas';
```

```
procedure pgm_$set_severity(in severity_level: integer);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pgm.ins.ftn'
```

```
integer*2 severity_level
```

```
call pgm_$set_severity(severity_level)
```

## DESCRIPTION

**Pgm\_\$set\_severity** sets the severity level for the calling process to *severity\_level*. Every invoked program returns a severity level to its invoking process, and by default the severity level is **pgm\_\$ok**.

## Choosing an Exit Severity Level

The following are examples of appropriate choices for an exit severity level:

- **Pgm\_\$true** or **pgm\_\$false** would be returned by a program that compared two values or tested some other condition.
- **Pgm\_\$warning** would be returned by a file deleting program when its target file didn't exist.
- **Pgm\_\$error** would be returned by a compiler if the input program contained a correctable syntax error, but the object was correct.
- **Pgm\_\$output\_invalid** would be returned by a compiler if an error in the input program resulted in an incorrect object.
- **Pgm\_\$internal\_fatal** would be returned by a program that could not proceed because its data was corrupted.
- **Pgm\_\$program\_faulted** would be returned by a program that received a fault and wished to inform the invoking program without resignaling the fault.

*severity\_level*

The exit severity level to return to the invoking process. Specify only one of the following values:

**pgm\_\$ok**

The program completed successfully and performed the requested action.

**pgm\_\$true**

The program completed successfully. Its purpose was to test a condition, and the value of that condition was true.

**pgm\_\$false**

The program completed successfully. Its purpose was to test a condition, and the value of that condition was false.

**pgm\_\$warning**

The program completed successfully and performed the requested action. However, an unusual (but nonfatal) condition was detected.

**pgm\_\$error**

The program could not perform the requested action because of syntactic or semantic errors in the input, though the output is sound.

**pgm\_\$output\_invalid**

The program could not perform the requested action because of syntactic or semantic errors in the input, and the output is not sound.

**pgm\_\$internal\_fatal**

The program detected an internal fatal error and ceased processing. The state of the output is not defined.

**pgm\_\$program\_faulted**

The program detected and handled a fault.

If *severity\_level* is set greater than **pgm\_\$max\_severity**, the status returned to the invoking process will be just **pgm\_\$max\_severity**.

**NOTES**

C programs that use **pgm\_\$set\_severity** must be bound with the **-entry** option to **/com/bind** or the **-e** option to **/bin/ld** to specify the start routine manually (normally "main"). The best alternative to **pgm\_\$set\_severity** is to use a UNIX **exit(2)** or a **return** to return a program status.

pm

---

**Contents**

intro .....	PM-1
pm_\$get_home_txt .....	PM-2
pm_\$get_sid_txt .....	PM-4

the 1990s, the number of people with a mental health problem has increased by 50% (Mental Health Act 1983, 1990).

There are a number of reasons for this increase. One of the reasons is that the definition of mental health problems has become broader. In the past, people with mental health problems were often those who were severely mentally ill and who were in need of hospital care. However, in the 1990s, the definition of mental health problems has become broader to include people who are less severely mentally ill and who are not in need of hospital care. This has led to an increase in the number of people who are classified as having a mental health problem.

Another reason for the increase is that there has been a change in the way that mental health problems are diagnosed. In the past, mental health problems were often diagnosed by a doctor or a psychiatrist. However, in the 1990s, there has been a move towards diagnosis by a range of professionals, including nurses, psychologists, and social workers. This has led to an increase in the number of people who are diagnosed with a mental health problem.

A third reason for the increase is that there has been a change in the way that mental health problems are treated. In the past, people with mental health problems were often treated with medication and hospital care. However, in the 1990s, there has been a move towards treatment with a range of services, including community care, day care, and self-help. This has led to an increase in the number of people who are treated for a mental health problem.

There are a number of implications of the increase in the number of people with a mental health problem. One of the implications is that there is a need for more services to be available to people with mental health problems. This includes more community care services, day care services, and self-help services. It also includes more services for people who are severely mentally ill and who are in need of hospital care.

Another implication of the increase is that there is a need for more research to be done into mental health problems. This includes research into the causes of mental health problems, the diagnosis of mental health problems, and the treatment of mental health problems. It also includes research into the social and cultural context of mental health problems.

There are a number of ways in which the increase in the number of people with a mental health problem can be addressed. One way is to increase the availability of services to people with mental health problems. This can be done by increasing the number of community care services, day care services, and self-help services. It can also be done by increasing the number of services for people who are severely mentally ill and who are in need of hospital care.

**NAME**

intro – the Domain/OS Process Manager

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/pm.h>
```

**SYNOPSIS (Pascal)**

```
%include 'sys/ins/base.ins.pas';
%include 'sys/ins/pm.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include 'sys/ins/base.ins.ftn'
%include 'sys/ins/pm.ins.ftn'
```

**DESCRIPTION**

The `pm_$` calls supply information from the environment of the calling process.

`Pm_$get_home_txt` reports the home directory of the calling process, and `pm_$get_sid_txt` reports the Subject Identifier (SID) of the calling process.

**Variable****`pm_$errout`**

A per-process external variable whose value is the stream ID of the standard error output. Domain/OS supports two different object formats: the old object format (obj) and the new Common Object File Format (COFF) which use different standard stream IDs. `Pm_$errout` allows dynamically loaded code to use one model or the other depending on the object format.

**Data Type****`pm_$sidtext_t`**

An array for passing SIDs as strings.

## NAME

`pm_$get_home_txt` – get the home directory

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pm.h>
```

```
void pm_$get_home_txt(
    short &buffer_length,
    char *path_name,
    short *path_length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pm.ins.pas';
```

```
procedure pm_$get_home_txt
in buffer_length: integer;
out path_name: univ_name_$long_pname_t;
out path_length: integer);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pm.ins.ftn'
```

```
integer*2 buffer_length, path_length
character path_name*1023
```

```
call pm_$get_home_txt(buffer_length, path_name, path_length)
```

## DESCRIPTION

`Pm_$get_home_txt` supplies the home directory associated with the SID of the calling process in *path\_name*.

*buffer\_length*

The number of bytes in the buffer allocated to receive the home directory. `Pm_$get_home_txt` will not write more than *buffer\_length* characters into *path\_name*.

*path\_name*

The pathname of the home directory for the calling process. The home directory supplied in *path\_name* is the one associated with the SID of the calling process and is the same as the `HOME` variable in the caller's environment.

*path\_length*

The number of bytes in the pathname of the home directory. If the value supplied in *path\_length* equals or exceeds the value passed in *buffer\_length* then the pathname written in *path\_name* might be truncated.

PM\_\$GET\_HOME\_TXT

PM\_\$GET\_HOME\_TXT

**SEE ALSO**

name\_\$get\_ndir\_lc, name\_\$get\_wdir\_lc.

## NAME

`pm_$get_sid_txt` – get the SID

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/pm.h>
```

```
void pm_$get_sid_txt(
    short &buffer_length,
    char *sid,
    short *sid_length);
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/pm.ins.pas';
```

```
procedure pm_$get_sid_txt(
    in buffer_length: integer;
    out sid: univ pm $sidtext_t;
    out sid_length: integer);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/pm.ins.ftn'
```

```
integer*2 buffer_length, sid_length
character sid*140
```

```
call pm_$get_sid_txt(buffer_length, sid, sid_length)
```

## DESCRIPTION

`Pm_$get_sid_txt` supplies the SID for the calling process in *sid*. A SID is a key to the permissions granted to a process, and has the form *''PERSON.GROUP.ORGANIZATION''*, where *PERSON* is the login of the owner of the process, *GROUP* is the process owner's current group, and *ORGANIZATION* is the process owner's current organization or *''project.''*

*buffer\_length*

The number of bytes in the buffer allocated to receive the SID. `Pm_$get_sid_txt` will not write more than *buffer\_length* characters into *sid*.

*sid*

The SID for the calling process.

*sid\_length*

The number of bytes in the SID of the calling process. If the value supplied in *sid\_length* equals or exceeds the value passed in *buffer\_length*, then the SID string written in *sid* might be truncated.

---

**Contents**

intro .....	PRF-1
prf_\$config_file .....	PRF-12
prf_\$edit_job .....	PRF-13
prf_\$get_printers .....	PRF-15
prf_\$get_sites .....	PRF-17
prf_\$init .....	PRF-19
prf_\$inq_option .....	PRF-20
prf_\$name_print .....	PRF-22
prf_\$queue_file .....	PRF-24
prf_\$read_queue .....	PRF-26
prf_\$set_option .....	PRF-28
prf_\$signal_printer .....	PRF-30
prf_\$stream_print .....	PRF-32

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every receipt and invoice should be properly filed and indexed for easy retrieval. This is particularly crucial for businesses that deal with a large volume of transactions or those in highly regulated industries.

Next, the document outlines the various methods used to collect and analyze financial data. It covers traditional techniques such as manual bookkeeping and the use of spreadsheets, as well as more modern approaches involving data mining and artificial intelligence. The goal is to provide a comprehensive overview of the tools and techniques available to financial analysts and accountants.

The third section focuses on the challenges of data security and privacy. With the increasing reliance on digital systems, the risk of data breaches and cyberattacks has become a significant concern. The document provides a detailed analysis of common security vulnerabilities and offers practical advice on how to implement robust security measures to protect sensitive financial information.

Finally, the document concludes with a discussion on the future of financial data management. It explores emerging trends such as cloud computing, blockchain technology, and the integration of big data analytics. The author predicts that these technologies will continue to revolutionize the way financial data is collected, stored, and analyzed, leading to more efficient and accurate financial reporting.

**NAME**

intro – the Domain/OS print library

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
```

**DESCRIPTION**

The **prf\_\$** calls allow an application to use Domain/OS printing services.

Domain/OS supports a network-wide collection of print managers which route print jobs to appropriate print servers. The **prf\_\$get\_sites** and **prf\_\$get\_printers** system calls supply information about the printing resources currently available on a Domain internet.

**Starting Print Jobs**

A print job consists of a print file containing data to be printed, and an associated print request. A print request is a structured file containing a list of print option name/value pairs that control the result of a print operation. The print file can be spooled from a stream, a spooled copy of a file retained by a user, or even another file not on the print spool.

The data structure queued in a print request is allocated whenever a process accesses the print library, and initialized via the **prf\_\$init** call. Thereafter, a process can modify the results of a print job by changing print option values from their defaults. The print option names are strings, but the print option values may be specified as strings or as numeric values. When specified numerically the print option values have type **num\_var\_t**.

After setting options via **prf\_\$config\_file**, and **prf\_\$set\_option** a process can queue a print request with **prf\_\$name\_print**, **prf\_\$queue\_file**, or **prf\_\$stream\_print**.

Until the next call to **prf\_\$init**, the process may find out and alter the status of the print services relevant to the queued print job with **prf\_\$edit\_job**. The **prf\_\$read\_queue** and **prf\_\$signal\_printer** calls allow a process to find out what jobs are pending or in progress and delete or abort them.

**Print Options**

The following print options apply to all types of files:

**SEARCH\_DIR ON|OFF**

**ON** tells the print server to search through the working directories of all the active processes on the node for the files to be printed. **OFF**, the default, tells the print server to search only the current working directory of the calling process.

**COPIES** *n*

Print *n* copies of the file. One copy is printed by default.

**PRINTER** *name*

Print the file on the printer named *name*. A printer's name is defined in its configuration file. If this option is not specified, the print server uses the default printer name "P".

**SITE** *node\_entry\_dir*

Queue the print request on the node whose node entry directory is *node\_entry\_dir*.

**DELETE** ON|OFF

ON tells the print server to delete the print file after it has been printed. The default state is on; that is, if this option is not specified, the print server deletes the print file indicated by the print request after printing.

**USER\_NAME** *name*

Print *name* as the user name on the banner page of the printed file. If the SIGNAL option is set to ALARM, the print server also sends an alarm to the user specified in *name* when printing is complete. Therefore, *name* must be a valid log-in name when requesting an alarm from the print server.

**SIGNAL** ALARM|OFF

ALARM tells the print server to notify a user when printing is finished. The print server sends an alarm to the user specified with the USER\_NAME option if it is set, or to the current log-in name for the calling process if the USER\_NAME option is not set. This option's default value is OFF.

**BANNER** ON|OFF

ON tells the print server to print a banner page displaying a job name, log-in name, and system information. OFF, the default, tells the print server to not print a banner page.

**CONFIG\_FILE** *pathname*

Read print request options from *pathname*. If *pathname* is null, the print server reads print request options from the configuration file at `~/user_data/prf.db`.

**TEXT** Specifies text mode for printing ASCII files. TEXT is the default print mode.

**PLOT** ON|OFF

Use PLOT ON to print GPR bitmaps or to print from graphics metafiles. PLOT OFF is the default.

**TRANSPARENT** ON|OFF

ON tells the print server to pass the file directly to the printer driver routine with no processing by the print server; that is, the printer can

interpret the file directly. **OFF**, the default, tells the print server to do it's normal interpretation for the target printer.

**PAPER\_SIZE** A|B|LEGAL|A3|A4|A5|B4|B5

Use the specified page size on the Domain/Laser-26 printer.

Paper Size Codes	
size code	paper dimensions in inches (mm)
A	8.50 × 11.00 (218 × 282)
B	11.00 × 17.00 (282 × 436)
LEGAL	8.50 × 14.00 (218 × 359)
A3	11.69 × 16.54 (297 × 420)
A4	8.27 × 11.69 (210 × 297)
A5	5.38 × 8.27 (137 × 210)
B4	9.84 × 13.90 (257 × 364)
B5	5.93 × 9.89 (182 × 257)

This option is only meaningful for Domain/Laser-26 and APPLE Laser-Writer printers.

The following print options apply to text files only:

**MARGINS ON|OFF**

**ON** tells the print server to use the margin settings specified with the **TOP**, **BOTTOM**, **RIGHT**, and **LEFT** options. **OFF** tells the print server to ignore the **TOP**, **BOTTOM**, **RIGHT**, and **LEFT** options. The default state is **ON**.

**TOP** *n* The top margin is *n* inches, where *n* is a real number.

**BOTTOM** *n*

The bottom margin is *n* inches, where *n* is a real number.

**RIGHT** *n*

The right margin is *n* inches, where *n* is a real number.

**LEFT** *n*

The left margin is *n* inches, where *n* is a real number.

**HEADERS ON|OFF**

**ON** tells the print server to use the header and footer defined with the **HEAD\_STRING** and **FOOT\_STRING** options. **OFF** tells the print server to ignore the header and footer defined with the **HEAD\_STRING** and **FOOT\_STRING** options. The default state is **ON**.

**HEAD\_STRING** *left/center/right*

Use this option to define a page header. Specify the header as a single string of three components delimited with slashes (/). The resulting header has *left* printed flush against the left margin, *center* centered on the page, and *right* printed flush against the right margin. Components may be empty; that is, *"/* is a valid string and results in a blank header. The print server interprets the following special character sequences and substitutes the strings indicated when the characters appear in the header string:

Special Character	Substituted String
@	The escape character. Turns off any special interpretation of the following character. For instance, "@*" results in a "*" instead of a space.
#	The current page number with a leading and a trailing space.
%c	The current date.
!	The name of the file being printed.
&	The time and date the file being printed was last modified.
*	A space. Inserts a space in the header. Literal spaces are not permitted in header strings.

The header string `"Page*@##!/Printed*on**%"` will produce a header with the label `"Page #"` followed by the page number flush left, the file name centered, and the label `"Printed on "` followed by the current date flush right.

**FOOT\_STRING** *left/center/right*

Use this option to define a page footer. The format is the same as for **HEAD\_STRING**.

**FTN ON|OFF**

**ON** tells the print server to use FORTRAN forms control even if the

file does not have the FORTRAN carriage control flag. With **FTN ON**, the print server interprets the first character of each line as a FORTRAN carriage control character and doesn't print it, so the first character of every line will be lost if the file does not use FORTRAN carriage control. The default state is **OFF**.

**WRAP ON|OFF**

**WRAP ON** tells the print server to wrap any lines that exceed the right margin onto the next line. **WRAP OFF** tells the print server to truncate lines that exceed the right margin. The default state is **OFF**.

The following print options apply to plot files; that is, graphics metafiles and GPR bitmaps:

**RESOLUTION *n***

Print the file with a resolution as close to *n* dots per inch as possible, where *n* is an integer. This option applies only to plot files.

**WHITE\_SPACE *n***

Leave *n* inches of white space between plots, where *n* is a real number. The default is 3 inches, or `WHITE_SPACE 3`.

**BW\_REV ON|OFF**

**ON** tells the print server to reverse black and white values in bitmaps. The default state is **OFF**.

**MAGNIFICATION *n***

Use this option to specify a bitmap magnification value, where *n* is an integer in the range -1 to 16. **MAGNIFICATION -1** tells the print server to scale the bitmap to fill the available page space. **MAGNIFICATION 0** tells the print server to print a graphics metafile with a one-to-one scaling between the display and the printer. If the print file is a GPR bitmap, **MAGNIFICATION 0** is equivalent to **MAGNIFICATION 1**. A magnification value of 1 through 16 scales the bitmap by that value. Regions of the magnified bitmap that extend beyond the printer's page boundaries are clipped.

The following print options are for printers that support variable font and pitch sizes:

**PITCH *n***

Set the printer's pitch to *n* characters per inch. The following pitch settings are supported:

Printer Pitch Settings	
Printer	Pitch
Printronix	10
Spinwriter	12
IMAGEN	8.5, 10, 12, 15, 17.1
GE 3000	10, 12, 13.1, 16.7
Versatec	12
LaserWriter	1 to 100
Laser-26	1 to 100

**POINT *n***

Set the size of the font to *n* points.

**WEIGHT LIGHT|MEDIUM|BOLD**

Use this option to set the font weight for a GE 3000 printer; it is not valid with other printers. The default is **WEIGHT MEDIUM**.

**LQ ON|OFF**

**ON** tells the print server to print the document at letter quality. **OFF** tells the print server to print in draft mode. The default state is **OFF**. The **LQ** option is valid only for GE 3000 printers.

The following options are for printers that contain a POSTSCRIPT interpreter, such as the Domain/Laser-26 and APPLE LaserWriter.

**POSTSCRIPT ON|OFF**

**ON** tells the print server to invoke the printer's POSTSCRIPT interpreter while printing. **OFF**, the default, tells the print server to bypass the printer's POSTSCRIPT interpreter.

**COLUMNS *i*|*2***

Format text in the specified number of columns. This option is meaningful only to a POSTSCRIPT printer. **COLUMNS 1** is the default.

**LPI *n*** Set the printer's line spacing to *n* lines per inch, where *n* is a real number.

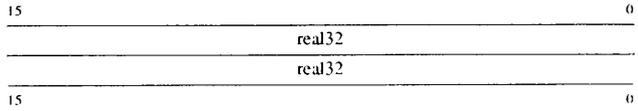
**ORIENTATION PORTRAIT|LANDSCAPE**

**PORTRAIT** tells the print server to print text lines or the x-axis of bitmaps parallel to the short edge of the page. **LANDSCAPE** tells the print server to print text lines or the x-axis of bitmaps perpendicular to the short leading edge. This option applies only to printers that include

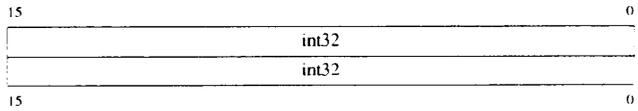
the POSTSCRIPT interpreter (i.e., Laser-26, LaserWriter, GENICOM, and V80 printers with the POSTSCRIPT decomposer software). The default state is **PORTRAIT**. Specifying this option will override any auto-rotation performed on bitmaps.

**Data Types****num\_var\_t**

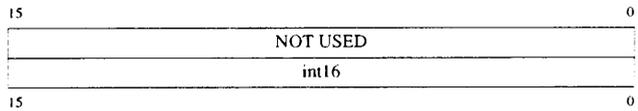
This is a variant record type for the numeric value of a print request option.



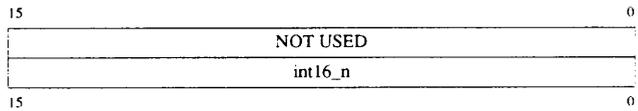
OR



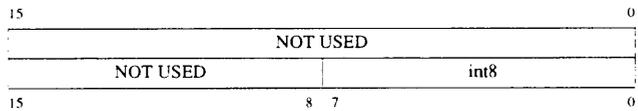
OR



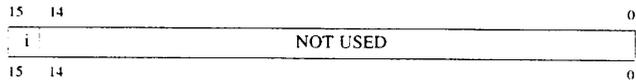
OR



OR



OR



- real32** A 32-bit real number.
- int32** A positive 31-bit number.
- int16** A positive 16-bit number.
- int16\_n** A positive or negative 16-bit number.
- int8** A positive 8-bit integer.
- int1** (labeled as "i" in the diagram) A boolean value.

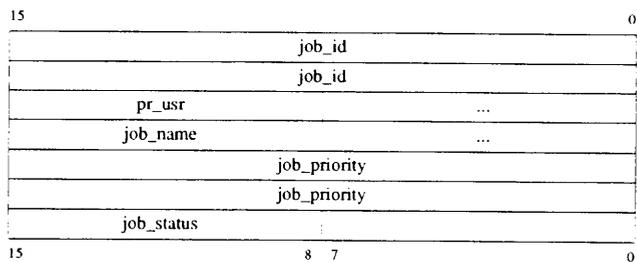
**prf\_\$edit\_job\_t**

An enumerated type for specifying the intended operation in the **prf\_\$edit\_job** call. It takes one of the following values:

- prf\_\$del\_job**  
Delete the job.
- prf\_\$prior**  
Alter the job's priority.
- prf\_\$time\_print**  
Postpone printing the job.

**prf\_\$job\_entry\_t**

A record for storing print job information.



- job\_id** An integer label for the job.
- pr\_usr** The user name or printer name for the job. It has type **prf\_\$name\_t**.
- job\_name**  
The name of the job. It has type **prf\_\$name\_t**.
- job\_priority**  
The priority of the job.
- job\_status**  
The status of the job. It has type **prf\_\$name\_t**.

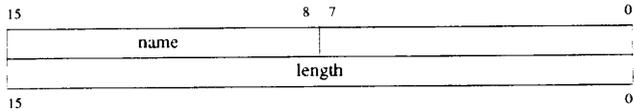
**prf\_\$job\_status\_t**

An enumerated type for specifying the status of a print job. It takes one of the following values:

- prf\_\$prntg**  
The job is printing.
- prf\_\$idle**  
The job is idle and waiting to print.
- prf\_\$susp**  
The job is suspended awaiting a continue signal.

**prf\_\$name\_t**

A record for passing string arguments to **prf\_\$** calls.



- name** The string. It may be up to 32 bytes long.
- length** The number of bytes in *name*.

**prf\_\$pr\_sig\_t**

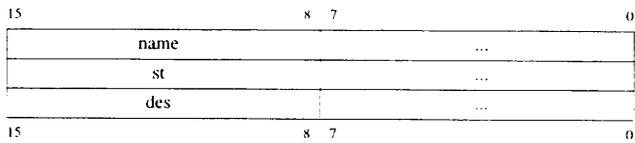
An enumerated type for specifying the signal to send with **prf\_\$signal\_printer**. It takes one of the following values:

- prf\_\$abort\_job**  
Abort the job.
- prf\_\$suspend**  
Suspend the job.

**prf\_\$continue**  
Continue the job.

**prf\_\$printer\_array\_t**  
An array of type **prf\_\$printer\_t**.

**prf\_\$printer\_t**  
A record type for holding information on a printer.



*name* The name of the printer. It has type **prf\_\$name\_t**.

*st* The status of the printer. It has type **prf\_\$name\_t**.

*des* A description of the printer. It has type **prf\_\$name\_t**.

**prf\_\$read\_opt\_t**  
An enumerated type for restricting the information returned by **prf\_\$read\_queue**. The type can currently take on only one possible value:

**prf\_\$read\_by\_printer**  
Return a list of jobs by printer.

**NOTES**

APPLE and LaserWriter are registered trademarks of Apple Computer, Inc. GENICOM is a trademark of Genicom corporation. IMAGEN is a registered trademark of the IMAGEN Corporation. POSTSCRIPT is a registered trademark of Adobe Systems. Spinwriter is a registered trademark of NEC. Versatec is a trademark of Versatec, Inc.

## NAME

`prf_$config_file` – set print options from a file

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>

void prf_$config_file(
    char *name,
    short &name_length,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$config_file(
    in name: name_$pname_t;
    in name_length: integer;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 name_length
integer*4 status
character name*1024
```

```
call prf_$config_file(name, name_length, status)
```

## DESCRIPTION

This call reads print request options from the file at *name* into the current print data structure.

*name* The full pathname of the print configuration file.

*name\_length*  
The number of bytes in *name*.

*status* The completion status.

## SEE ALSO

`prf_$inq_option`, `prf_$set_option`.

## NAME

`prf_$edit_job` – edit a print job at the current site

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>

void prf_$edit_job(
    long &job_id,
    prf_$edit_job_t &edit_op,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';

procedure prf_$edit_job(
    in job_id: integer32;
    in edit_op: prf_$edit_job_t;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'

integer*4 job_id, status
integer*2 edit_op

call prf_$edit_job(job_id, edit_op, status)
```

## DESCRIPTION

This call performs the operation specified by *edit\_op* on the job specified by *job\_id*. The *job\_id* must refer to an entry in the queue at the current site.

*job\_id* The ID of the print job to edit. The *job\_id* is the first member of a record of type `prf_$job_entry_t`.

*edit\_op* The edit operation to perform on the job. Specify one of the following:

- `prf_$del_job`  
Delete the job.
- `prf_$prior`  
Alter the job's priority.
- `prf_$time_print`  
Postpone printing the job.

*status* The completion status.

## NOTES

Use `prf_$read_queue` to get an array of job entries, of type `prf_$job_entry_t`, from the

PRF\_\$EDIT\_JOB

PRF\_\$EDIT\_JOB

queue at the current print site.

SEE ALSO

prf\_\$delete\_entry.

## NAME

`prf_$get_printers` – get a list of printers at the current site

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_$get_printers(
    prf_$name_t *site,
    long *entry,
    long &max_printers,
    long *ret_printers,
    prf_$printer_t *printer_list,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$get_printers(
    in site: univ prf_$name_t;
    var entry: integer32;
    in max_printers: integer32;
    out ret_printers: integer32;
    out printer_list: univ prf_$printer_array_t;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 n_printers, example
parameter (n_printers = 32, example = 3)
```

```
integer*4 entry, max_printers, ret_printers
character printer_list(n_printers)*102
```

```
equivalence (printer_name, printer_list(example))
equivalence (printer_name_length, printer_list(example))(33:34)
equivalence (printer_status, printer_list(example))(35:66)
equivalence (printer_status_length, printer_list(example))(67:68)
equivalence (printer_description, printer_list(example))(69:100)
equivalence (printer_description_length, printer_list(example))(101:102)
```

```
call prf_$get_printers(site, entry, max_printers,
& ret_printers, printer_list, status)
```

## DESCRIPTION

This call supplies up to *max\_printers* printer names from the current site in the *printer\_list* argument. The *entry* argument passed to `prf_$get_printers` controls which portion of the current site's printer list is supplied in *printer\_list*.

*site* The print site to list printers for. If *site* is null, then all sites on the network are polled.

*entry* An index into the printer list at the current site. *Index* specifies the position in the printer list of the first printer requested when `prf_$get_printers` is called; it contains the position in the list of the next printer to request when `prf_$get_printers` returns.

When calling `prf_$get_printers`, an *entry* of 1 will cause `prf_$get_printers` to supply the first *max\_printers* printer names from the site's printer list in *printer\_list*. Then when `prf_$get_printers` returns, *entry* will be the position of the next unread printer name in the list.

If the difference between the call and return values of *entry* is less than *max\_printers*, then *printer\_list* contains the name of the last printer in the site list. If the difference is equal to *max\_printers*, there may be more printers at the site; that is, *printer\_list* may not contain the name of the last printer in the printer list. If the printer list is exhausted when `prf_$get_printers` returns, *entry* contains the position of the last printer, or, more simply, just the number of printers at the current site.

*max\_printers*

The maximum number of printers that `prf_$get_printers` should supply. The value of *max\_printers* depends on the size of the buffer allocated to receive the *printer\_list* argument.

*ret\_printers*

The number of printer descriptions supplied in *printer\_list*.

*printer\_list*

An array of type `prf_$name_t` containing a list of printer descriptions. Each element of the array contains a printer's name, its description, and its status.

*status* The completion status.

## SEE ALSO

`prf_$get_sites`, `prf_$read_queue`.

## NAME

`prf_$get_sites` – get a list of print sites

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>

void prf_$get_sites(
    long *index,
    long &max_sites,
    long *ret_sites,
    prf_$name_t *site_list,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$get_sites(
    var index: integer32;
    in max_sites: integer32;
    out ret_sites: integer32;
    out site_list: univ prf_$name_array_t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 n_sites, example
parameter (n_sites = 32, example = 3)

integer*4 index, max_sites, ret_sites, status
character site_list(n_sites)*34

integer*2 site_location_length
character site_location*32

equivalence (site_location, site_list(example))
equivalence (site_location_length, site_list(example) (33:34))

call prf_$get_sites(index, max_sites, ret_sites, site_list, status)
```

## DESCRIPTION

This call supplies the locations of up to `max_sites` print sites on the network. The `index` argument passed to `prf_$get_sites` controls which portion of the network site list is supplied in `site_list`.

*index* An index into the list of sites supplied by `prf_$get_sites`. *Index* specifies the position in the network site list of the first site requested when `prf_$get_sites` is called; it contains the position in the list of the next site to request when `prf_$get_sites` returns.

When calling `prf_$get_sites`, an *index* of 1 will cause `prf_$get_sites` to supply the first *max\_sites* site locations. Then when `prf_$get_sites` returns, *index* will be the position of the next unread site location in the list.

If the difference between the call and return values of *index* is less than *max\_sites*, then *site\_list* contains the location of the last site in the network site list. If the difference is equal to *max\_sites*, there may be more sites on the network; that is, *site\_list* may not contain the location of the last print site in the site list. If the site list is exhausted when `prf_$get_sites` returns, *index* contains the position of the last site, or, more simply, just the number of sites on the network.

*max\_sites*

The maximum number of sites that `prf_$get_sites` should supply. The value of *max\_sites* depends on the size of the buffer allocated to receive the *site\_list* argument.

*ret\_sites*

The number of sites supplied in *site\_list*.

*site\_list* An array of type `prf_$name_t` containing a list of sites. Each element of the array is a string containing the site's location followed by the number of bytes in the string.

*status* The completion status.

SEE ALSO

`prf_$get_printers`.

## NAME

`prf_$init` – initialize print request options

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_$init(
    ios_$id_t &stream_id,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$init(
    in stream_id: ios_$id_t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 stream_id
integer*4 status

call prf_$init(stream_id, status)
```

## DESCRIPTION

This call resets the options in the current print data structure to their default values and establishes a stream to use for error and status information returned from the print server.

*stream\_id*

The stream to write error and status messages to.

*status*

The completion status.

## SEE ALSO

`prf_$config_file`, `prf_$inq_option`, `prf_$set_option`.

## NAME

`prf_$inq_option` – get a print request option

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_$inq_option(
    char *option,
    short *option_length,
    num_var_t *number_value,
    char *string_value,
    short *string_length,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$inq_option(
    var option: univ string;
    var option_length: integer;
    out number_value: num_var_t;
    out string_value: univ string;
    out string_length: integer;
    out status: status_$t)
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 opt_len
parameter (opt_len = 120)
```

```
integer*2 option_length, string_length
integer*4 number_value, status
character option*(opt_len), string_value*256
```

```
call prf_$inq_option(option, option_length, number_value,
& string_value, string_length, status)
```

## DESCRIPTION

This call gets the string and numeric value for the option specified in *option*.

*option* The name of the option inquired about.

*option\_length*

The number of bytes in *option*.

*number\_value*

The numeric value of the option requested in *option*.

*string\_value*

The string value of the option requested in *option*.

*string\_length*

The number of bytes in *string\_value*.

*status*

The completion status.

**SEE ALSO**

prf\_\$config\_file, prf\_\$set\_option.

## NAME

`prf_$name_print` – spool a file for printing

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_$name_print(
    char *pathname,
    short &pathname_length,
    char *queue,
    short &queue_length,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$name_print(
    in pathname: univ name_$name_t;
    in pathname_length: integer ;
    out queue: name_$name_t;
    out queue_length: integer;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 pathname_length, queue_length
integer*4 status
character pathname*1024, queue*1024
```

```
call prf_$name_print(pathname, pathname_length, queue,
& queue_length, status)
```

## DESCRIPTION

This call copies the file at *pathname* to the print spool and queues a print request for the job using the option values in the current print data structure. `Prf_$name_print` supplies the pathname of the queued print request in *queue* when it returns.

*pathname*

The pathname of the file to print.

*pathname\_length*

The number of bytes in *pathname*.

*queue*

The full pathname of the queued print request.

*queue\_length*

The number of bytes in *queue*.

*status* The completion status.

#### NOTES

Both `prf_$name_print` and `prf_$queue_file` queue a print request for a file, and thus initiate a print job to print a file. The difference between them is that `prf_$name_print` copies the file to a print spool, and the resulting print request prints from the spooled copy. Consequently, when the print job terminates and the print server deletes the file indicated by the print request, it is the spooled copy that is deleted — not the original file named in the `prf_$name_print` call. `Prf_$queue_file` doesn't copy the file to a print spool; so, when the print job terminates and the print server deletes the file indicated by the print request, the original file named in the `prf_$queue_file` call is deleted.

Therefore, while setting `DELETE OFF` for a print request queued by `prf_$queue_file` is a good safety precaution for preserving the original file, setting `DELETE OFF` for a print request queued by `prf_$name_print` will leave orphaned files on the print spool and is not generally a good idea.

#### SEE ALSO

`prf_$stream_print`.

## NAME

`prf_queue_file` – queue a print request

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_queue_file(
    char *pathname,
    short &pathname_length,
    char *queue,
    short &queue_length,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_queue_pathname(
    in pathname: univ name $pname_t;
    in pathname_length: integer;
    out queue: name $pname_t;
    out queue_length: integer;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 pathname_length, queue_length
integer*4 status
character pathname*1024, queue*1024
```

```
call prf_queue_file(pathname, pathname_length, queue,
& queue_length, status)
```

## DESCRIPTION

This call queues a print request for the file at *pathname*. It does not copy the file at *pathname* to the print spool, so the print server will delete the original file itself rather than a spooled copy when the print job completes unless the **DELETE** option is set to **OFF**. When `prf_queue_file` returns, it supplies the pathname of the queued print request in *queue*.

*pathname*

The pathname of the file to print.

*pathname\_length*

The number of bytes in *pathname*.

*queue* The pathname of the queued print request.

*queue\_length*  
The number of bytes in *queue*.

*status* The completion status.

#### NOTES

Both `prf_$name_print` and `prf_$queue_file` queue a print request, and thus initiate a print job to print a file. The difference between them is that `prf_$name_print` copies the file to a print spool and the resulting print request prints from the spooled copy. Consequently, when the print job terminates and the print server deletes the file indicated by the print request, it is the spooled copy that is deleted — not the original file named in the `prf_$name_print` call. `Prf_$queue_file` doesn't copy the file to a print spool, so when the print job terminates and the print server deletes the file indicated by the print request, the original file named in the `prf_$queue_file` call is deleted.

Therefore, while setting `DELETE OFF` for a print request queued by `prf_$queue_file` is a good safety precaution for preserving the original file, setting `DELETE OFF` for a print request queued by `prf_$name_print` will leave orphaned files on the print spool and is not generally a good idea.

#### SEE ALSO

`prf_$stream_print`.

## NAME

`prf_$read_queue` – get a list of print jobs in the queue

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_$read_queue(
    prf_$read_opt_t &printer_flag,
    long *index,
    long &max_jobs,
    long *ret_jobs,
    prf_$job_entry_t *job_list,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$read_queue(
    in printer_flag: prf_$read_opt_t;
    var index: integer32;
    in max_jobs: integer32;
    out ret_jobs: integer32;
    out job_list: univ prf_$job_array_t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 n_jobs, example
parameter (n_jobs = 32, example = 3)
```

```
integer*4 printer_flag, index, status
integer*4 max_jobs, ret_jobs
integer*2 job_list(39, n_jobs)
```

```
integer*4 job_id, job_priority
integer*2 print_user_length, job_name_length, job_status
character print_user*32, job_name*32
```

```
equivalence (job_id, job_list(1, example))
equivalence (print_user, job_list(3, example))
equivalence (print_user_length, job_list(19, example))
equivalence (job_name, job_list(20, example))
equivalence (job_name_length, job_list(36, example))
```

**equivalence** (*job\_priority*, *job\_list*(37, *example*))  
**equivalence** (*job\_status*, *job\_list*(39, *example*))

call **prf\_\$read\_queue**(*printer\_flag*, *index*, *max\_jobs*,  
 & *ret\_jobs*, *job\_list*, *status*)

#### DESCRIPTION

This call supplies a list of queued jobs at the current site.

##### *printer\_flag*

An enumerated value that specifies how jobs will be listed in *job\_list*. Currently, the only legal value for *printer\_flag* is **prf\_\$read\_by\_printer**, which causes **prf\_\$read\_queue** to return only those jobs queued for the printer specified in the current print data structure.

##### *index*

An index into the jobs in the current queue. *Index* specifies the position in the queue of the first job requested when **prf\_\$read\_queue** is called; it contains the position in the queue of the next job to request when **prf\_\$read\_queue** returns.

When calling **prf\_\$read\_queue**, an *index* of 1 will cause **prf\_\$read\_queue** to supply the first *max\_jobs* jobs from the queue in *job\_list*. Then when **prf\_\$read\_queue** returns, *index* will be the position of the next unread job in the queue.

If the difference between the call and return values of *index* is less than *max\_jobs*, then *job\_list* contains the last job in the queue. If the difference is equal to *max\_jobs*, there may be more jobs in the queue; that is, *job\_list* may not contain the last job in the queue. If the job list is exhausted when **prf\_\$read\_queue** returns, *index* contains the position of the last job, or, more simply, just the number of jobs queued at the current site.

##### *max\_jobs*

The maximum number of jobs that **prf\_\$read\_queue** should supply. The value of *max\_jobs* depends on the size of the buffer allocated to receive the *job\_list* argument.

*ret\_jobs* The number of jobs supplied in *job\_list*.

*job\_list* An array of type **prf\_\$job\_entry\_t** containing a list of jobs from the current queue.

*status* The completion status.

#### SEE ALSO

**prf\_\$edit\_job**, **prf\_\$get\_printers**, **prf\_\$get\_sites**.

## NAME

`prf_$set_option` – set an option in a print request

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>

void prf_$set_option(
    char *option,
    short &option_length,
    num_var_t &string_value,
    char *p_str,
    short &string_length,
    boolean &string_encoding,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';

procedure prf_$set_option(
    in option: univ string;
    in option_length: integer;
    in number_value: num_var_t;
    in string_value: univ string;
    in string_length: integer;
    in string_encoding: boolean;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'

integer*2 opt_len
parameter (opt_len = 120)

integer*2 option_length, string_length
integer*4 number_value, status
character option*(opt_len), string_value*256
logical string_encoding

call prf_$set_option(option, option_length, number_value, string_value,
& string_length, string_encoding, status)
```

## DESCRIPTION

This call sets the print option specified by *option* in the current print data structure.

*option* The name of the option to set.

*option\_length*

The number of bytes in *option*.

*number\_value*

The numeric value to set *option* to.

*string\_value*

The string value to set *option* to.

*string\_length*

The number of bytes in *string\_value*.

*string\_encoding*

A Boolean argument indicating whether the value of *option* is encoded as a string or number. If *string\_encoding* is **true**, the value of *option* is set from *string\_value*. If **false**, the value of *option* is set from *number\_value*.

*status* The completion status.

**SEE ALSO**

`prf_$config_file`, `prf_$inq_option`.

## NAME

**prf\_\$signal\_printer** – signal a print job

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>

void prf_$signal_printer(
    prf_$pr_sig_t &signal,
    prf_$name_t *printer_name,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';

procedure prf_$signal_printer(
    in signal: prf_$pr_sig_t;
    in printer_name: univ prf_$name_t;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'

integer*4 status
integer*2 signal, length
character printer_name*34, name*32

equivalence (name, printer_name(1:32))
equivalence (length, printer_name(33:34))

call prf_$signal_printer(signal, printer_name, status)
```

## DESCRIPTION

This call sends the signal specified in *signal* to the printer named *printer\_name* at the current site.

*signal* Specify one of the following:

- prf\_\$abort\_job**  
Abort the job.
- prf\_\$suspend**  
Suspend the job.
- prf\_\$continue**  
Continue the job.

*printer\_name*  
The name of the printer.

PRF\_\$SIGNAL\_PRINTER

PRF\_\$SIGNAL\_PRINTER

*status* The completion status.

**SEE ALSO**

prf\_\$delete\_entry, prf\_\$edit\_job, prf\_\$read\_queue.

## NAME

**prf\_\$\$stream\_print** – print from a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/prf.h>
```

```
void prf_$$stream_print(
    stream_$$id_t &stream_id,
    char *queue,
    short &queue_length,
    status_$$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/prf.ins.pas';
```

```
procedure prf_$$stream_print(
    in stream_id: stream_$$id_t;
    out queue: name_$$name_t;
    out queue_length: integer;
    out status: status_$$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
integer*2 stream_id, queue_length
integer*4 status
character queue*1024
```

```
call prf_$$stream_print(stream_id, queue, queue_length, status)
```

## DESCRIPTION

This call copies data from the stream specified by *stream\_id* to a file on the print spool, then queues a print request for the spooled file. When **prf\_\$\$stream\_print** returns, it supplies the pathname of the resulting print request in *queue*.

*stream\_id*

The ID of the stream to print from. Everything on the stream up to an end-of-file marker (EOF) is spooled.

*queue* The pathname of the resulting print request.

*queue\_length*

The number of bytes in *queue*.

*status* The completion status.

## SEE ALSO

**prf\_\$\$name\_print**, **prf\_\$\$queue\_file**.

# proc1

---

## Contents

intro .....	PROC1-1
proc1_\$get_cput .....	PROC1-2

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, accounts payable, and accounts receivable. It also outlines the procedures for recording these transactions, including the use of double-entry bookkeeping to ensure that the books are balanced.

The second part of the document focuses on the analysis of the recorded data. It explains how to calculate key financial ratios and metrics, such as the gross profit margin, operating profit, and return on investment. These calculations are essential for understanding the company's financial performance and identifying areas for improvement. The document also discusses the importance of comparing the company's performance against industry benchmarks and historical data to provide context for the results.

Finally, the document addresses the issue of financial reporting and the preparation of financial statements. It provides a step-by-step guide to the preparation of the balance sheet, income statement, and cash flow statement. It also discusses the requirements for these statements, including the need for accuracy and transparency. The document concludes by emphasizing the role of financial reporting in providing stakeholders with the information they need to make informed decisions about the company's future.

**NAME**

intro – the Level 1 Process Manager

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/procl.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/procl.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/procl.ins.ftn'
```

**DESCRIPTION**

There is one call in the Level 1 Process Manager interface, `procl_$get_cput`, which reports the total CPU time used by the calling process.

**Constants**

`procl_$n_user_processes`  
The maximum number of user processes per node.

**SEE ALSO**

`time_$intro`.

## NAME

`proc1_$get_cput` – get elapsed CPU time

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/proc1.h>
```

```
void proc1_$get_cput(
    time_$clock_t *clock_value)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/proc1.ins.pas';
```

```
procedure proc1_$get_cput(
    out clock_value: time_$clock_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/proc1.ins.ftn'
```

```
integer*2 clock_value(3), clockh
integer*4 clockl
```

```
equivalence (clockh, clock_value(1)), (clockl, clock_value(2))
```

```
call proc1_$get_cput(clock_value)
```

## DESCRIPTION

`Proc1_$get_cput` supplies the elapsed CPU time for the calling process in *clock\_value*.  
*clock\_value*

The amount of CPU time used by the calling process since its creation. *Clock\_value* is a 48-bit integer counter of the number of 4-microsecond system clock periods that have elapsed while the calling process was running in the CPU. It includes time spent by the operating system in servicing the process, but not time the process spent waiting for I/O or suspended.

# proc2

---

## Contents

intro .....	PROC2-1
proc2_\$get_info .....	PROC2-4
proc2_\$list .....	PROC2-6
proc2_\$who_am_i .....	PROC2-7

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, accounts payable, and accounts receivable. It also outlines the procedures for recording these transactions, including the use of journals and ledgers. The second part of the document focuses on the reconciliation process, which is essential for identifying and correcting errors. It describes how to compare the company's records with bank statements and other external sources to ensure that the numbers match. The document also discusses the importance of regular audits and the role of internal controls in preventing fraud and mismanagement. Finally, the document concludes with a summary of the key points and a call to action for the management team to implement the recommended practices.

**NAME**

intro – the Level 2 Process Manager

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/proc2.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/proc2.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/proc2.ins.ftn'
```

**DESCRIPTION**

The `proc_$` calls supply information about Level 2 context of processes running on the local node.

**Data Types**

`proc2_$info_t`

A record type for passing process information. The following diagram illustrates its format:

15		0
	stack_uid	
	state	
	Not Used	
	usr	
	Not Used	
	upc	
	upc	
	usp	
	usp	
	usb	
	usb	
	cpu_total	
	cpu_total	
	cpu_total	
	Not Used	
	priority	
15		0

- stack\_uid**     The user stack UID.
- stack\_base**   The base address of the user stack.
- state**         The process state, a value of type `proc2_$state_t`.
- usr**            The user status register.
- upc**            The user program counter.
- usp**            The user stack pointer.
- usb**            The user stack base pointer.
- cpu\_total**     The cumulative CPU time used.

**priority**

The process priority.

**proc2\_\$uid\_list\_t**

An array of type **uid\_t** for passing up to **proc1\_\$n\_user\_processes** process UIDs.

**proc2\_\$state\_t**

A set type for describing the state of a user process. It can take any combination of the following predefined values:

**proc2\_\$waiting**

The process is waiting.

**proc2\_\$suspended**

The process is suspended.

**proc2\_\$susp\_pending**

The process suspension is pending.

**proc2\_\$bound**

The process is bound.

**Errors****proc2\_\$bad\_stack\_base**

Bad stack base.

**proc2\_\$is\_current**

The completion status of **proc2\_\$get\_info** when passed the UID for the calling process. It does not indicate a failure.

**proc2\_\$not\_level\_2**

The completion status of **proc2\_\$get\_info** when passed the UID for a process that is not a user process.

**proc2\_\$uid\_not\_found**

Process not found.

## NAME

proc2\_\$get\_info – get level 2 process information

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/proc2.h>
```

```
void proc2_$get_info(
    uid_t &process_uid,
    proc2_$info_t *process_info,
    pinteger &buffer_length,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/proc2.ins.pas';
```

```
procedure proc2_$get_info(
    in process_uid: uid_t;
    out process_info: univ proc2_$info_t;
    in buffer_length: pinteger;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/proc2.ins.ftn'
```

```
integer*4 process_uid(2), status
integer*2 buffer_length, process_info(18)
```

```
integer*4 stack_uid(2), stack_base, upc, usp, usb, cpu_total_low
integer*2 state, usr, cpu_total(3), cpu_total_high, priority
```

```
equivalence (stack_uid, process_info(1))
equivalence (stack_base, process_info(5))
equivalence (state, process_info(7))
equivalence (usr, process_info(8))
equivalence (upc, process_info(9))
equivalence (usp, process_info(11))
equivalence (usb, process_info(13))
equivalence (cpu_total, process_info(15))
equivalence (priority, process_info(18))
equivalence (cpu_total_high, cpu_total(1))
equivalence (cpu_total_low, cpu_total(2))
```

```
call proc2_$get_info(process_uid, process_info, buffer_length, status)
```

## DESCRIPTION

**Proc2\_\$get\_info** supplies information about the local process specified by *process\_uid*.

*process\_uid*

The UID of a process.

*process\_info*

Information about the process specified in *process\_uid*. If *process\_uid* is the UID of the calling process, **proc\_\$get\_info** supplies only the stack UID and stack base pointer in *process\_info*.

*buffer\_length*

The number of bytes allocated to receive *process\_info*. **Proc2\_\$get\_info** will not write more than *buffer\_length* bytes of information into *process\_info*.

*status*

The completion status. In addition to **status\_\$ok**, **proc2\_\$get\_info** can return successfully with a *status* of **proc2\_\$is\_current**, indicating that the UID passed in *process\_uid* is for the calling process and that, therefore, the only valid information supplied in *process\_info* is the stack UID and the stack base pointer. If *process\_uid* does not correspond to a process running on the local node, **proc2\_\$get\_info** fails with a *status* of **proc2\_\$uid\_not\_found**.

## NOTES

**Proc2\_\$who\_am\_i** and **proc2\_\$list** supply process UIDs.

## SEE ALSO

**proc1\_\$cpu\_time**.

## NAME

`proc2_$list` – list level 2 process UIDs

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/proc2.h>
```

```
void proc2_$list(
    uid_t *uid_list,
    pinteger &max_num_uids,
    pinteger *num_uids)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/proc2.ins.pas';
```

```
procedure proc2_$list(
    out uid_list: univ proc2_$uid_list_t;
    in max_num_uids: pinteger;
    out num_uids: pinteger);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/proc2.ins.ftn'
```

```
integer*4 uid_list(48)
integer*2 max_num_uids, num_uids

call proc2_$list(uid_list, max_num_uids, num_uids)
```

## DESCRIPTION

`Proc2_$list` supplies a list of up to `max_num_uids` level 2 processes (user processes) running on the local node in `uid_list`.

`uid_list` A list of the up to `max_num_uids` UIDs for level 2 processes running on the local node.

`max_num_uids`

The maximum number of UIDs that `proc2_$list` should supply. `Proc2_$list` will not write more than `max_num_uids` UIDs into `uid_list`.

`num_uids`

The number of active level 2 processes on the local node. If `num_uids` is greater than `max_num_uids`, then only a partial list was written into `uid_list`.

## SEE ALSO

`proc2_$who_am_i`.

**NAME**

`proc2_$who_am_i` – get the UID of the calling process

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/proc2.h>
```

```
void proc2_$who_am_i(
    uid_t *my_uid)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/proc2.ins.pas';
```

```
procedure proc2_$who_am_i(
    out my_uid: uid_t);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/proc2.ins.ftn'
```

```
integer*4 my_uid(2)
```

```
call proc2_$who_am_i(my_uid)
```

**DESCRIPTION**

`Proc2_$who_am_i` supplies the UID of the calling process in *my\_uid*.

*my\_uid* The UID of the calling process.

**SEE ALSO**

`proc2_$list`.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, accounts payable, and accounts receivable. It also outlines the procedures for recording these transactions, including the use of double-entry bookkeeping to ensure that the books balance.

The second part of the document focuses on the analysis of the recorded data. It explains how to calculate key financial ratios and metrics, such as the gross profit margin, operating profit, and return on investment. These calculations are essential for understanding the company's financial performance and identifying areas for improvement. The document also discusses the importance of comparing the company's performance against industry benchmarks and historical data to provide context for the results.

The final part of the document addresses the reporting requirements for the financial data. It outlines the format and content of the financial statements, including the balance sheet, income statement, and cash flow statement. It also discusses the importance of providing clear and concise explanations for any significant fluctuations in the data. The document concludes by emphasizing the need for transparency and accountability in financial reporting, and the role of the accounting department in ensuring that all information is accurate and reliable.

---

**Contents**

intro .....	RWS-1
rws_\$alloc .....	RWS-3
rws_\$alloc_heap_pool .....	RWS-4
rws_\$alloc_rw_pool .....	RWS-6
rws_\$release_heap_pool .....	RWS-8



**NAME**

intro – dynamic storage allocation

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/rws.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/rws.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/rws.ins.ftn'
```

**DESCRIPTION**

The `rws_$` calls allow programs to allocate storage dynamically. There are two types of storage supported by the system. "Heap storage" can be allocated and released by user processes, whereas "read/write storage" can only be allocated by a user process and is released only when the allocating process terminates. Heap storage allows more control over storage, but requires slightly more system overhead to maintain.

**Data Types****`rws_$pool_t`**

An enumerated type for specifying the type of storage to allocate. It can assume one of the following values:

**`rws_$std_pool`**

Storage allocated from the standard pool is private to the allocating process and is released during a UNIX `exec(2)` call. A pointer to storage allocated from the standard pool is not valid in another process.

**`rws_$stream_tm_pool`**

Storage allocated from the stream pool is private to the allocating process and is retained across a UNIX `exec(2)` call. A pointer to storage allocated from the standard pool is not valid in another process.

**`rws_$global_pool`**

Storage allocated from the global pool is available to all processes running on a single node. A pointer to storage allocated from the global pool may be used by all local processes to access a common storage area.

**Errors****`rws_$bad_free`**

Attempted to free storage allocated from a different pool.

**`rws_$level_failure`**

Program level information was corrupted by a user process.

**rws\_\$non\_existent\_pool**

Invalid pool specified.

**rws\_\$no\_space**

Not enough address space or disk space.

**rws\_\$not\_heap\_entry**

Argument to **rws\_\$release\_heap** did not refer to storage allocated with **rws\_\$alloc\_heap**.

**rws\_\$scribbled\_over**

Heap process information was corrupted by a user process.

**rws\_\$wrong\_level**

Attempted to release standard or stream pool storage that was allocated by a program at a lower program level.

## NAME

`rws_$alloc` – allocate storage

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/rws.h>
```

```
void rws_$alloc(
    long &storage_size,
    void **storage_ptr)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/rws.ins.pas';
```

```
procedure rws_$alloc(
    in storage_size: integer32;
    out storage_ptr: univ_ptr);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/rws.ins.ftn'
```

```
integer*4 nwords
parameter (nwords = 5000)
```

```
integer*4 storage_size, storage_ptr
integer*2 dummy(nwords)
pointer /storage_ptr/ dummy
```

```
call rws_$alloc(storage_size, storage_ptr)
```

## DESCRIPTION

`Rws_$alloc` allocates *storage\_size* bytes of storage to the calling process and supplies a pointer to it in *storage\_ptr*.

All storage allocated with `rws_$alloc` remains allocated until process termination. It cannot be released back to the system.

*storage\_size*

The number of bytes of storage to allocate.

*storage\_ptr*

The address of the new storage space. If *storage\_ptr* is NULL, nil, or 0, then `rws_$alloc` could not allocate the requested storage.

## NOTES

`Rws_$alloc_heap_pool` and `rws_$alloc_rw_pool` provide more control over storage allocation than `rws_$alloc`.

## NAME

`rws_$alloc_heap_pool` – allocate heap storage

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/rws.h>
```

```
void *rws_$alloc_heap_pool(
    rws_$pool_t &alloc_pool,
    long &storage_size)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/rws.ins.pas';
```

```
function rws_$alloc_heap_pool(
    in alloc_pool: rws_$pool_t;
    in storage_size: rws_$word_aligned_long): univ_ptr;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/rws.ins.ftn'
```

```
integer*4 nwords
parameter (nwords = 5000)
```

```
integer*4 storage_size, storage_ptr
integer*2 alloc_pool, dummy(nwords)
pointer (/storage_ptr/ dummy)
```

```
storage_ptr = rws_$alloc_heap_pool(alloc_pool, storage_size)
```

## DESCRIPTION

`Rws_$alloc_heap_pool` allocates `storage_size` bytes of heap storage from the pool specified by `alloc_pool` and returns a pointer to it. It returns NULL, nil, or 0, when it cannot allocate the requested storage.

*alloc\_pool*

The pool from which the storage will be allocated. It can assume one of the following values:

`rws_$std_pool`

Storage allocated from the standard pool is private to the allocating process and is released during a UNIX `exec(2)` call. A pointer to storage allocated from the standard pool is not valid in another process.

`rws_$stream_tm_pool`

Storage allocated from the stream pool is private to the allocating process and is retained across a UNIX `exec(2)` call. A pointer to storage

RWS\_\$ALLOC\_HEAP\_POOL

RWS\_\$ALLOC\_HEAP\_POOL

allocated from the standard pool is not valid in another process.

**rws\_\$global\_pool**

Storage allocated from the global pool is available to all processes running on a single node. A pointer to storage allocated from the global pool may be used by all local processes to access a common storage area.

*storage\_size*

The number of bytes of storage needed.

**NOTES**

Unlike storage allocated with `rws_$alloc_rw_pool`, storage allocated with `rws_$alloc_heap_pool` can be released back to the system with `rws_$release_heap_pool`, though the storage requires slightly more overhead.

**SEE ALSO**

`rws_$alloc`.

## NAME

`rws_$alloc_rw_pool` – allocate read/write storage from a pool

## SYNOPSIS (C)

```
#include <apollo/base.h>
```

```
#include <apollo/rws.h>
```

```
void *rws_$alloc_rw_pool(
    rws_$pool_t &alloc_pool,
    long &storage_size)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
```

```
%include '/sys/ins/rws.ins.pas';
```

```
function rws_$alloc_rw_pool(
    in alloc_pool: rws_$pool_t;
    in storage_size: rws_$word_aligned_long): univ_ptr;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
%include '/sys/ins/rws.ins.ftn'
```

```
integer*4 nwords
parameter (nwords = 5000)
```

```
integer*4 storage_size, storage_ptr
integer*2 alloc_pool, dummy(nwords)
pointer /storage_ptr/ dummy
```

```
storage_ptr = rws_$alloc_rw_pool(alloc_pool, storage_size)
```

## DESCRIPTION

`Rws_$alloc_rw_pool` allocates *storage\_size* bytes of read/write storage from the pool specified by *alloc\_pool* and returns a pointer to it. It returns NULL, nil, or 0, when it cannot allocate the requested storage.

*alloc\_pool*

The pool from which the storage will be allocated.

*rws\_\$std\_pool*

Storage allocated from the standard pool is private to the allocating process and is released during a UNIX `exec(2)` call. A pointer to storage allocated from the standard pool is not valid in another process.

*rws\_\$stream\_tm\_pool*

Storage allocated from the stream pool is private to the allocating process and is retained across a UNIX `exec(2)` call. A pointer to storage allocated from the standard pool is not valid in another process.

RWS\_\$ALLOC\_RW\_POOL

RWS\_\$ALLOC\_RW\_POOL

**rws\_\$global\_pool**

Storage allocated from the global pool is available to all processes running on a single node. A pointer to storage allocated from the global pool may be used by all local processes to access a common storage area.

*storage\_size*

The number of bytes of storage needed.

**NOTES**

Unlike storage allocated with **rws\_\$alloc\_heap\_pool** storage allocated with **rws\_\$alloc\_rw\_pool** cannot be released back to the system.

**SEE ALSO**

**rws\_\$alloc.**

## NAME

`rws_$release_heap_pool` – release heap storage

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/rws.h>

void rws_$release_heap_pool(
    void *&storage_ptr,
    rws_$pool_t &alloc_pool,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/rws.ins.pas';

procedure rws_$release_heap_pool(
    in storage_ptr: univ_ptr;
    in alloc_pool: rws_$pool_t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/rws.ins.ftn'

integer*4 nwords
parameter (nwords = 5000)

integer*4 storage_ptr, status
integer*2 alloc_pool, dummy(nwords)
pointer (/storage_ptr/) dummy

call rws_$release_heap_pool(storage_ptr, alloc_pool, status)
```

## DESCRIPTION

`Rws_$release_heap_pool` releases the storage at `storage_ptr` back to the system.

`storage_ptr`

A pointer to storage allocated from the heap with `rws_$alloc_heap_pool`.

`alloc_pool`

The pool from which the storage was allocated. Choose one of the following values:

`rws_$std_pool`

Storage allocated from the standard pool is private to the allocating process and is released during a UNIX `exec(2)` call. A pointer to storage allocated from the standard pool is not valid in another process.

RWS\_\$RELEASE\_HEAP\_POOL

RWS\_\$RELEASE\_HEAP\_POOL

**rws\_\$stream\_fm\_pool**

Storage allocated from the stream pool is private to the allocating process and is retained across a UNIX exec(2) call. A pointer to storage allocated from the standard pool is not valid in another process.

**rws\_\$global\_pool**

Storage allocated from the global pool is available to all processes running on a single node. A pointer to storage allocated from the global pool may be used by all local processes to access a common storage area.

*status* The completion status.

**NOTES**

Unlike storage allocated with `rws_$alloc_rw_pool`, storage allocated with `rws_$alloc_heap_pool` can be released back to the system with `rws_$release_heap_pool`.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, employee salaries, and utility bills. It also outlines the proper procedures for recording these transactions, including the use of double-entry bookkeeping and the importance of regular reconciliations.

The second part of the document focuses on the analysis of the recorded data. It explains how to calculate key financial ratios and metrics, such as the gross profit margin and the current ratio. These calculations are essential for understanding the company's financial health and performance. The document also discusses the importance of comparing the company's results to industry benchmarks and historical data to identify trends and areas for improvement. Finally, it provides a summary of the findings and offers recommendations for future actions based on the analysis.

sio

---

**Contents**

intro .....	SIO-1
sio_\$control .....	SIO-10
sio_\$inquire .....	SIO-17

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial statements. This includes not only sales and purchases but also expenses, income, and any other financial activity.

The second part of the document provides a detailed breakdown of the accounting process. It starts with the identification of the accounting cycle, which consists of eight steps: identifying the accounting cycle, analyzing and journalizing the transactions, posting to the ledger, determining debits and credits, preparing a trial balance, adjusting the entries, preparing financial statements, and closing the books.

The third part of the document discusses the importance of the trial balance. It explains that the trial balance is a statement that lists all the accounts and their balances at a specific point in time. It is used to check the accuracy of the accounting records and to ensure that the debits equal the credits.

The fourth part of the document discusses the importance of the financial statements. It explains that the financial statements are a summary of the company's financial performance over a period of time. They include the income statement, the balance sheet, and the statement of cash flows.

The fifth part of the document discusses the importance of the closing process. It explains that the closing process is the final step in the accounting cycle, and it involves transferring the balances of the temporary accounts to the permanent accounts.

**NAME**

intro – controlling serial I/O lines

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/sio.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/sio.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/sio.ins.ftn'
```

**DESCRIPTION**

The calls are the program interface to the serial I/O (SIO) lines. They can control and monitor the state of an RS-232 port, and modify the stream connections to it. The file system interface to the SIO lines are named `/dev/sion` where *n* is some integer.

**Constants**

**sio\_\$50** A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$75** A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$110**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$134**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$150**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$300**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$600**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$1200**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$2000**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$2400**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$3600**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$4800**

A baud rate value for the `sio_$speed` and `sio_$speed_force` options.

**sio\_\$7200**

A baud rate value for the **sio\_\$speed** and **sio\_\$speed\_force** options.

**sio\_\$9600**

A baud rate value for the **sio\_\$speed** and **sio\_\$speed\_force** options.

**sio\_\$19200**

A baud rate value for the **sio\_\$speed** and **sio\_\$speed\_force** options.

**sio\_\$even\_parity**

A possible value for the **sio\_\$parity** option, denoting even parity.

**sio\_\$max\_line**

The highest SIO line number supported.

**sio\_\$no\_parity**

A possible value for the **sio\_\$parity** option, denoting no parity.

**sio\_\$odd\_parity**

A possible value for the **sio\_\$parity** option, denoting odd parity.

**sio\_\$stop\_1**

A stop bit value for the **sio\_\$stop\_bits** option.

**sio\_\$stop\_1\_point\_5**

A stop bit value for the **sio\_\$stop\_bits** option.

**sio\_\$stop\_2**

A stop bit value for the **sio\_\$stop\_bits** option.

**sio\_\$5bpc**

A possible value for the **sio\_\$bits\_per\_char** option, denoting five bits per character.

**sio\_\$6bpc**

A possible value for the **sio\_\$bits\_per\_char** option, denoting six bits per character.

**sio\_\$7bpc**

A possible value for the **sio\_\$bits\_per\_char** option, denoting seven bits per character.

**sio\_\$8bpc**

A possible value for the **sio\_\$bits\_per\_char** option, denoting eight bits per character.

**Data Types****sio\_\$err\_enables\_t**

A small set of enabled SIO errors. It can assume a combination of the following values:

**sio\_\$check\_parity**

Check for received parity errors.

- sio\_\$check\_framing**  
Check for received framing errors.
- sio\_\$check\_dcd\_change**  
Check for when the Data Carrier Detect (DCD) line changes state.
- sio\_\$check\_cts\_change**  
Check for when the Clear To Send (CTS) line changes state.
- sio\_\$err\_possibilities\_t**  
A small set of errors to report during read calls on an SIO line. It can assume a combination of the following values:
  - sio\_\$check\_parity**  
Report parity errors.
  - sio\_\$check\_framing**  
Report framing errors. Framing errors are reported by default.
  - sio\_\$check\_dcd\_change**  
Report DCD line state changes.
  - sio\_\$check\_cts\_change**  
Report CTS line state changes.
- sio\_\$line\_t**  
An SIO line number between 0 and **sio\_\$max\_line**.
- sio\_\$opt\_t**  
An enumerated type for specifying an SIO option. It takes one of the following values:
  - sio\_\$bits\_per\_char**  
The number of bits per character. This option can have one of the following values:
    - sio\_\$5bpc**  
Five bits per character.
    - sio\_\$6bpc**  
Six bits per character.
    - sio\_\$7bpc**  
Seven bits per character.
    - sio\_\$8bpc**  
Eight bits per character.
  - sio\_\$bp\_enable**  
This option is a Boolean value that enables or disables interpretation of control codes from a bit pad. If **true**, bit pad control code interpretation is enabled. If **false**, bit pad control code interpretation is disabled. The default value is **false**.

**sio\_\$cts** This option is a Boolean value that reflects the state of the Clear to Send (CTS) line. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$cts\_enable**  
This option is a Boolean value that specifies whether the CTS line will be used to inhibit transmission. If **true**, then transmission is inhibited when CTS is false. If **false**, the CTS line does not inhibit transmission.

**sio\_\$dcd**  
This option is a Boolean value that reflects the state of the Data Carrier Detect (DCD) line. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$dcd\_enable**  
This option is a Boolean value that specifies whether a DCD line transition will generate a **fault\_\$stop** fault. If **true**, then a DCD line transition from **true** to **false** generates a fault. If **false**, a DCD line transition does not generate a fault.

**sio\_\$drain\_out**  
This option is a Boolean value that specifies whether write calls on the SIO line should wait until all characters in the output buffer are transmitted before returning. If **true**, the write call waits. If **false**, the write call doesn't wait. The default is **false**.

**sio\_\$dtr**  
This option is a Boolean value that sets the state of the Data Terminal Ready (DTR) line.

**sio\_\$eofchr**  
The value of this option is the end-of-file (EOF) character. It takes a character value, and the default is CTRL/Z.

**sio\_\$erase**  
The value of this option is the erase character. It takes a character value, and the default is CTRL/H.

**sio\_\$err\_enable**  
The value of this option is a small set of type **sio\_\$err\_possibilities\_t** that defines the errors that can be reported by read calls on an SIO line. It can be any combination of the following values:

**sio\_\$check\_parity**  
Report parity errors.

**sio\_\$check\_framing**  
Report framing errors. Framing errors are reported by default.

**sio\_\$check\_dcd\_change**  
Report DCD line state changes.

**sio\_\$check\_cts\_change**  
Report CTS line state changes.

**sio\_\$flush\_in**  
This option is a Boolean value that specifies whether to flush the input buffer of an SIO line. If **true**, the input buffer is flushed. If **false**, the input buffer is not flushed. The default is **false**.

**sio\_\$flush\_out**  
This option is a Boolean value that specifies whether to flush the output buffer of an SIO line. If **true**, the output buffer is flushed. If **false**, the output buffer is not flushed. The default is **false**.

**sio\_\$host\_sync**  
This option is a Boolean value that enables or disables host synchronization. If **true**, the node sends XOFF (CTRL/S) when its input buffer is full, and sends XON (CTRL/Q) when it is again ready to receive data. If **false**, the node does not use XON/XOFF synchronization. The default is **true**.

**sio\_\$shup\_close**  
This option is a Boolean value that specifies whether the SIO line should "hang up" when the last stream on it is closed. If **true**, the node drops the Data Terminal Ready (DTR) line for approximately three-fourths of a second when the last stream on it is closed. If **false**, the node does not hang up.

**sio\_\$input\_sync**  
This option is a Boolean value that enables or disables input synchronization. If **true**, the node honors incoming XON and XOFF requests. If **false**, the node ignores incoming XON/XOFF signals. The default is **false**.

**sio\_\$int\_enable**  
This option is a Boolean value that enables or disables interrupts from the calling process. If **true**, the SIO line honors interrupts from the calling process. If **false**, the SIO line ignores interrupts from the calling process.

**sio\_\$intchr**  
The value of this option is the interrupt character. It takes a character value, and the default is CTRL/C.

**sio\_\$kill**  
The value of this option is the kill character. It takes a character value, and the default is CTRL/X.

**sio\_\$line**  
This value of this option is the SIO line number, an integer from 0 to **sio\_\$max\_line**. The value of this option cannot be changed with

**sio\_\$control.**

**sio\_\$nlc\_delay**

The value of this option is the time delay in milliseconds following transmission of a line feed character, to allow for carriage motion or scrolling time, and so on. The default is 0.

**sio\_\$no\_NL**

This option is a Boolean value that specifies whether a newline character is transmitted as a carriage-return line-feed sequence or as a raw newline character. If **true**, newlines are transmitted as is. If **false**, newlines are transmitted as carriage-return line-feed sequences. The default is **false**.

**sio\_\$no\_echo**

This option is a Boolean value that specifies whether input characters are echoed back in the output. If **true**, input characters are not echoed. If **false**, input characters are echoed back to the sender. The default is **false**.

**sio\_\$parity**

The value of this option is the parity used by the SIO line. It can take any one of the following values:

**sio\_\$even\_parity**

Even parity.

**sio\_\$no\_parity**

No parity.

**sio\_\$odd\_parity**

Odd parity.

If the parity is even or odd, then one bit is added to each character to enforce the chosen parity. If there are fewer than eight bits per character, the parity bit is delivered with the data in raw mode, but is stripped in cooked mode.

**sio\_\$quit\_enable**

This option is a Boolean value that specifies whether the node will respond to a quit character or break condition by interrupting the calling process. If **true**, a quit or break generates a quit fault. If **false**, a quit or break is ignored. The default is **false**. (In raw mode, break and quit are the only ways to generate a quit fault.)

**sio\_\$quitchr**

The value of this option is the quit character. It takes a character value, and the default is CTRL/.

**sio\_\$raw**

This option is a Boolean value that determines whether the line is in

raw or cooked mode. If **true**, the line is in raw mode and each read returns all bytes received since the last call without any interpretation. If **false**, the line is in cooked mode and control codes are interpreted. The default value is **false**.

The input buffer is flushed whenever **sio\_\$raw** changes value.

**sio\_\$raw\_nl**

This option is a Boolean value that specifies whether a newline character is transmitted as a carriage-return line-feed sequence or as a raw newline character when the line is in raw mode. If **true**, newlines are transmitted as is while in raw mode. If **false**, newlines are transmitted as carriage-return line-feed sequences while in raw mode. The default is **false**.

**sio\_\$rts** This option is a Boolean value that determines the state of the Request To Send (RTS) line. The default is **true**.

**sio\_\$rts\_enable**

This option is a Boolean value that enables or disables the RTS line. If **true**, the RTS line is used with the CTS line for data flow control. For flow control to work, the CTS line must also be enabled. If **false**, no RTS/CTS flow control is provided. The default is **false**.

**sio\_\$send\_break**

The value of this option is the duration of a break condition on the SIO line, in milliseconds. A reasonable value for the **sio\_\$send\_break** option is 200. A value other than 0 causes a break condition.

**sio\_\$speed**

The value of this option is the baud rate of the SIO line. It can have any of the following values: **sio\_\$50**, **sio\_\$75**, **sio\_\$110**, **sio\_\$134**, **sio\_\$150**, **sio\_\$300**, **sio\_\$600**, **sio\_\$1200**, **sio\_\$2000**, **sio\_\$2400**, **sio\_\$3600**, **sio\_\$4800**, **sio\_\$7200**, **sio\_\$9600**, or **sio\_\$19200**. The default is **sio\_\$9600**.

**sio\_\$speed\_force**

The value of this option is the baud rate of a partnered SIO line. The baud rate is forced to the value of this option, even if it is incompatible with the rate of the line's partner. If the value of **sio\_\$speed\_force** is incompatible with the speed of the partnered line, the partnered line is forced to the default speed of **sio\_\$9600**. This option can have any of the following values: **sio\_\$50**, **sio\_\$75**, **sio\_\$110**, **sio\_\$134**, **sio\_\$150**, **sio\_\$300**, **sio\_\$600**, **sio\_\$1200**, **sio\_\$2000**, **sio\_\$2400**, **sio\_\$3600**, **sio\_\$4800**, **sio\_\$7200**, **sio\_\$9600**, or **sio\_\$19200**.

**sio\_\$stop\_bits**

The value of this option is the number of stop bits used by the SIO line. It can have any of the following values: **sio\_\$stop\_1**,

`sio_$stop_1_point_5`, or `sio_$stop_2`. The default is `sio_$stop_1`.

**`sio_$susp_enable`**

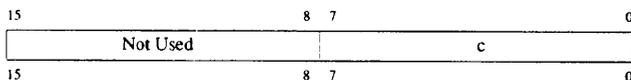
This option is a Boolean value that enables or disables suspend signals from the calling process. If `true`, the SIO line honors suspend signals (faults). If `false`, the SIO line ignores suspend signals (faults).

**`sio_$suspchr`**

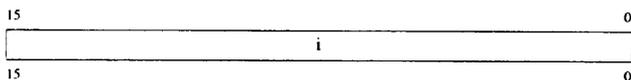
The value of this option is the suspend character. It takes a character value, and the default is CTRL/P.

**`sio_$value_t`**

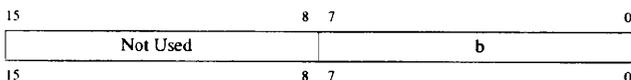
A variant record type for passing the values of SIO options. The diagram below illustrates the `sio_$value_t` data type:



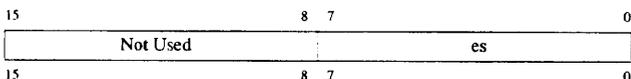
OR



OR



OR



- c** A character value.
- i** A 2-byte integer value.
- b** A Boolean value.
- es** A set of type `sio_$err_enables_t`.

**Errors****sio\_\$bad\_option**

The option passed is not a valid SIO option.

**sio\_\$illegal\_strid**

The stream ID passed in the `sio_$` call is not open on an SIO line.

**sio\_\$incompatible\_speed**

Attempted to set the speed for a partnered line that is incompatible with the speed of the line's partner.

## NAME

`sio_$control` – set serial line options

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/sio.h>

void sio_$control(
    stream_id_t &stream_id,
    sio_$opt_t &sio_option,
    sio_$value_t &option_value,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/sio.ins.pas';

procedure sio_$control
    in stream_id: stream_id_t;
    in sio_option: sio_$opt_t;
    in option_value: univ_sio_$value_t;
    out status: status_t;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/sio.ins.ftn'

integer*2 nchar
parameter (nchar = 256)

integer*4 status, option_value
integer*2 stream_id, sio_option, i2_value
logical l_value
character c_value*(nchar)

equivalence (i2_value, option_value)
equivalence (l_value, option_value)
equivalence (c_value, option_value)

call sio_$control(stream_id, sio_option, option_value, status)
```

## DESCRIPTION

`Sio_$control` sets the option specified by `sio_option` to the value passed in `option_value` for the SIO line open on `stream_id`.

The hardware configuration for some machine types is such that certain SIO lines are "partnered" with each other. The following is a list of machine types and the SIO lines that are partnered on them.

Serial I/O Lines	
Machine Type	Partnered Lines
DN300	1,2
DSP80	1,2
DN460	0,1
DN660	2,3
DN550	1,2
DN3000	No Partners
DN4000	No Partners

A characteristic of partnered lines is that some baud rates are incompatible. The following lists show the baud rates that are incompatible for partnered lines.

Incompatible Baud Rates	
A Rates	B Rates
sio_\$50	sio_\$75
sio_\$7200	sio_\$150
	sio_\$2000
	sio_\$19200

If one partner is set to a baud rate in the A list, an attempt to set the other partner to a baud rate in the B list (using the `sio_$speed` option) will result in the error `sio_$incompatible_speed`. The same is true for the reverse (having a partnered line set to a rate in the B list and attempting to set its partner to a rate in the A list). Speeds that don't appear in the table are compatible with any speed.

`Sio_$control` can force a line's speed to one that is incompatible with its partner by using the `sio_$speed_force` option; however, it will also change the speed of the partnered line to `sio_$9600`, which is compatible with any speed.

*stream\_id*

The stream ID of a stream open on a serial line.

*sio\_option*

An SIO line option. (The value of `sio_$cts`, `sio_$dcd`, and `sio_$line` cannot be changed with `sio_$control`. They are only valid when calling `sio_$inquire`.) Specify only one of the following values:

`sio_$bits_per_char`

The number of bits per character. This option can have one of the following values:

`sio_$5bpc`

Five bits per character.

**sio\_\$6bpc**  
Six bits per character.

**sio\_\$7bpc**  
Seven bits per character.

**sio\_\$8bpc**  
Eight bits per character.

**sio\_\$bp\_enable**  
This option is a Boolean value that enables or disables interpretation of control codes from a bit pad. If **true**, bit pad control code interpretation is enabled. If **false**, bit pad control code interpretation is disabled. The default value is **false**.

**sio\_\$cts** This option is a Boolean value that reflects the state of the Clear to Send (CTS) line. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$cts\_enable**  
This option is a Boolean value that specifies whether the CTS line will be used to inhibit transmission. If **true**, then transmission is inhibited when CTS is **false**. If **false**, the CTS line does not inhibit transmission.

**sio\_\$dcd**  
This option is a Boolean value that reflects the state of the Data Carrier Detect (DCD) line. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$dcd\_enable**  
This option is a Boolean value that specifies whether a DCD line transition will generate a **fault\_\$stop** fault. If **true**, then a DCD line transition from **true** to **false** generates a fault. If **false**, a DCD line transition does not generate a fault.

**sio\_\$drain\_out**  
This option is a Boolean value that specifies whether write calls on the SIO line should wait until all characters in the output buffer are transmitted before returning. If **true**, the write call waits. If **false**, the write call doesn't wait. The default is **false**.

**sio\_\$dtr**  
This option is a Boolean value that sets the state of the Data Terminal Ready (DTR) line.

**sio\_\$eofchr**  
The value of this option is the end-of-file (EOF) character. It takes a character value, and the default is CTRL/Z.

**sio\_\$erase**  
The value of this option is the erase character. It takes a character

value, and the default is CTRL/H.

**sio\_\$err\_enable**

The value of this option is a small set of type `sio_$err_possibilities_t` that defines the errors that can be reported by read calls on an SIO line. It can be any combination of the following values:

**sio\_\$check\_parity**

Report parity errors.

**sio\_\$check\_framing**

Report framing errors. Framing errors are reported by default.

**sio\_\$check\_dcd\_change**

Report DCD line state changes.

**sio\_\$check\_cts\_change**

Report CTS line state changes.

**sio\_\$flush\_in**

This option is a Boolean value that specifies whether to flush the input buffer of an SIO line. If `true`, the input buffer is flushed. If `false`, the input buffer is not flushed. The default is `false`.

**sio\_\$flush\_out**

This option is a Boolean value that specifies whether to flush the output buffer of an SIO line. If `true`, the output buffer is flushed. If `false`, the output buffer is not flushed. The default is `false`.

**sio\_\$host\_sync**

This option is a Boolean value that enables or disables host synchronization. If `true`, the node sends XOFF (CTRL/S) when its input buffer is full, and sends XON (CTRL/Q) when it is again ready to receive data. If `false`, the node does not use XON/XOFF synchronization. The default is `true`.

**sio\_\$hup\_close**

This option is a Boolean value that specifies whether the SIO line should "hang up" when the last stream on it is closed. If `true`, the node drops the Data Terminal Ready (DTR) line for approximately three-fourths of a second when the last stream on it is closed. If `false`, the node does not hang up.

**sio\_\$input\_sync**

This option is a Boolean value that enables or disables input synchronization. If `true`, the node honors incoming XON and XOFF requests. If `false`, the node ignores incoming XON/XOFF signals. The default is `false`.

**sio\_\$int\_enable**

This option is a Boolean value that enables or disables interrupts from

the calling process. If **true**, the SIO line honors interrupts from the calling process. If **false**, the SIO line ignores interrupts from the calling process.

**sio\_\$intr**

The value of this option is the interrupt character. It takes a character value, and the default is CTRL/C.

**sio\_\$kill**

The value of this option is the kill character. It takes a character value, and the default is CTRL/X.

**sio\_\$line**

This value of this option is the SIO line number, an integer from 0 to **sio\_\$max\_line**. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$nlc\_delay**

The value of this option is the time delay in milliseconds following transmission of a line feed character, to allow for carriage motion or scrolling time, and so on. The default is 0.

**sio\_\$no\_NL**

This option is a Boolean value that specifies whether a newline character is transmitted as a carriage-return line-feed sequence or as a raw newline character. If **true**, newlines are transmitted as is. If **false**, newlines are transmitted as carriage-return line-feed sequences. The default is **false**.

**sio\_\$no\_echo**

This option is a Boolean value that specifies whether input characters are echoed back in the output. If **true**, input characters are not echoed. If **false**, input characters are echoed back to the sender. The default is **false**.

**sio\_\$parity**

The value of this option is the parity used by the SIO line. It can take any one of the following values:

**sio\_\$even\_parity**

Even parity.

**sio\_\$no\_parity**

No parity.

**sio\_\$odd\_parity**

Odd parity.

If the parity is even or odd, then one bit is added to each character to enforce the chosen parity. If there are fewer than eight bits per character, the parity bit is delivered with the data in raw mode, but is stripped

in cooked mode.

**sio\_\$quit\_enable**

This option is a Boolean value that specifies whether the node will respond to a quit character or break condition by interrupting the calling process. If **true**, a quit or break generates a quit fault. If **false**, a quit or break is ignored. The default is **false**. (In raw mode, break and quit are the only ways to generate a quit fault.)

**sio\_\$quitchr**

The value of this option is the quit character. It takes a character value, and the default is CTRL/].

**sio\_\$raw**

This option is a Boolean value that determines whether the line is in raw or cooked mode. If **true**, the line is in raw mode and each read returns all bytes received since the last call without any interpretation. If **false**, the line is in cooked mode and control codes are interpreted. The default value is **false**.

The input buffer is flushed whenever **sio\_\$raw** changes value.

**sio\_\$raw\_nl**

This option is a Boolean value that specifies whether a newline character is transmitted as a carriage-return line-feed sequence or as a raw newline character when the line is in raw mode. If **true**, newlines are transmitted as is while in raw mode. If **false**, newlines are transmitted as carriage-return line-feed sequences while in raw mode. The default is **false**.

**sio\_\$rts** This option is a Boolean value that determines the state of the Request To Send (RTS) line. The default is **true**.

**sio\_\$rts\_enable**

This option is a Boolean value that enables or disables the RTS line. If **true**, the RTS line is used with the CTS line for data flow control. For flow control to work, the CTS line must also be enabled. If **false**, no RTS/CTS flow control is provided. The default is **false**.

**sio\_\$send\_break**

The value of this option is the duration of a break condition on the SIO line, in milliseconds. A reasonable value for the **sio\_\$send\_break** option is 200. A value other than 0 causes a break condition.

**sio\_\$speed**

The value of this option is the baud rate of the SIO line. It can have any of the following values: **sio\_\$50**, **sio\_\$75**, **sio\_\$110**, **sio\_\$134**, **sio\_\$150**, **sio\_\$300**, **sio\_\$600**, **sio\_\$1200**, **sio\_\$2000**, **sio\_\$2400**, **sio\_\$3600**, **sio\_\$4800**, **sio\_\$7200**, **sio\_\$9600**, or **sio\_\$19200**. The default is **sio\_\$9600**.

**sio\_\$speed\_force**

The value of this option is the baud rate of a partnered SIO line. The baud rate is forced to the value of this option, even if it is incompatible with the rate of the line's partner. If the value of **sio\_\$speed\_force** is incompatible with the speed of the partnered line, the partnered line is forced to the default speed of **sio\_\$9600**. This option can have any of the following values: **sio\_\$50**, **sio\_\$75**, **sio\_\$110**, **sio\_\$134**, **sio\_\$150**, **sio\_\$300**, **sio\_\$600**, **sio\_\$1200**, **sio\_\$2000**, **sio\_\$2400**, **sio\_\$3600**, **sio\_\$4800**, **sio\_\$7200**, **sio\_\$9600**, or **sio\_\$19200**.

**sio\_\$stop\_bits**

The value of this option is the number of stop bits used by the SIO line. It can have any of the following values: **sio\_\$stop\_1**, **sio\_\$stop\_1\_point\_5**, or **sio\_\$stop\_2**. The default is **sio\_\$stop\_1**.

**sio\_\$susp\_enable**

This option is a Boolean value that enables or disables suspend signals from the calling process. If **true**, the SIO line honors suspend signals (faults). If **false**, the SIO line ignores suspend signals (faults).

**sio\_\$suspchr**

The value of this option is the suspend character. It takes a character value, and the default is CTRL/P.

*option\_value*

The new value for the option specified by *sio\_option*.

*status* The completion status. Possible values are:

**sio\_\$bad\_option**

The option passed is not a valid SIO option.

**sio\_\$illegal\_strid**

The stream ID passed in the **sio\_\$** call is not open on an SIO line.

**sio\_\$incompatible\_speed**

Attempted to set the speed for a partnered line that is incompatible with the speed of the line's partner.

## NAME

*sio\_\$inquire* – get serial line options

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/sio.h>
```

```
void sio_$inquire(
    stream_id_t &stream_id,
    sio_$opt_t &sio_option,
    sio_$value_t *option_value,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/sio.ins.pas';
```

```
procedure sio_$inquire(
    in stream_id: stream_id_t;
    in sio_option: sio_$opt_t;
    out option_value: univ sio_$value_t;
    out status: status_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/sio.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 256)
```

```
integer*4 status, option_value
integer*2 stream_id, sio_option, i2_value
logical l_value
character c_value*(nchar)
```

```
equivalence (i2_value, option_value)
equivalence (l_value, option_value)
equivalence (c_value, option_value)
```

```
call sio_$inquire(stream_id, sio_option, option_value, status)
```

## DESCRIPTION

*Sio\_\$inquire* supplies the value of the option specified by *sio\_option* for the SIO line open on *stream\_id*.

*stream\_id*

The stream ID of a stream open on a serial line.

*sio\_option*

An SIO line option. Specify only one of the following values:

**sio\_\$bits\_per\_char**

The number of bits per character. This option can have one of the following values:

**sio\_\$5bpc**

Five bits per character.

**sio\_\$6bpc**

Six bits per character.

**sio\_\$7bpc**

Seven bits per character.

**sio\_\$8bpc**

Eight bits per character.

**sio\_\$bp\_enable**

This option is a Boolean value that enables or disables interpretation of control codes from a bit pad. If **true**, bit pad control code interpretation is enabled. If **false**, bit pad control code interpretation is disabled. The default value is **false**.

**sio\_\$cts** This option is a Boolean value that reflects the state of the Clear to Send (CTS) line. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$cts\_enable**

This option is a Boolean value that specifies whether the CTS line will be used to inhibit transmission. If **true**, then transmission is inhibited when CTS is **false**. If **false**, the CTS line does not inhibit transmission.

**sio\_\$dcd**

This option is a Boolean value that reflects the state of the Data Carrier Detect (DCD) line. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$dcd\_enable**

This option is a Boolean value that specifies whether a DCD line transition will generate a **fault\_\$stop** fault. If **true**, then a DCD line transition from **true** to **false** generates a fault. If **false**, a DCD line transition does not generate a fault.

**sio\_\$drain\_out**

This option is a Boolean value that specifies whether write calls on the SIO line should wait until all characters in the output buffer are transmitted before returning. If **true**, the write call waits. If **false**, the write call doesn't wait. The default is **false**.

- sio\_\$dtr**  
This option is a Boolean value that sets the state of the Data Terminal Ready (DTR) line.
- sio\_\$eofchr**  
The value of this option is the end-of-file (EOF) character. It takes a character value, and the default is CTRL/Z.
- sio\_\$erase**  
The value of this option is the erase character. It takes a character value, and the default is CTRL/H.
- sio\_\$err\_enable**  
The value of this option is a small set of type `sio_$err_possibilities_t` that defines the errors that can be reported by read calls on an SIO line. It can be any combination of the following values:
- sio\_\$check\_parity**  
Report parity errors.
  - sio\_\$check\_framing**  
Report framing errors. Framing errors are reported by default.
  - sio\_\$check\_dcd\_change**  
Report DCD line state changes.
  - sio\_\$check\_cts\_change**  
Report CTS line state changes.
- sio\_\$flush\_in**  
This option is a Boolean value that specifies whether to flush the input buffer of an SIO line. If `true`, the input buffer is flushed. If `false`, the input buffer is not flushed. The default is `false`.
- sio\_\$flush\_out**  
This option is a Boolean value that specifies whether to flush the output buffer of an SIO line. If `true`, the output buffer is flushed. If `false`, the output buffer is not flushed. The default is `false`.
- sio\_\$host\_sync**  
This option is a Boolean value that enables or disables host synchronization. If `true`, the node sends XOFF (CTRL/S) when its input buffer is full, and sends XON (CTRL/Q) when it is again ready to receive data. If `false`, the node does not use XON/XOFF synchronization. The default is `true`.
- sio\_\$hup\_close**  
This option is a Boolean value that specifies whether the SIO line should "hang up" when the last stream on it is closed. If `true`, the node drops the Data Terminal Ready (DTR) line for approximately three-fourths of a second when the last stream on it is closed. If `false`,

the node does not hang up.

**sio\_\$input\_sync**

This option is a Boolean value that enables or disables input synchronization. If **true**, the node honors incoming XON and XOFF requests. If **false**, the node ignores incoming XON/XOFF signals. The default is **false**.

**sio\_\$int\_enable**

This option is a Boolean value that enables or disables interrupts from the calling process. If **true**, the SIO line honors interrupts from the calling process. If **false**, the SIO line ignores interrupts from the calling process.

**sio\_\$intchr**

The value of this option is the interrupt character. It takes a character value, and the default is CTRL/C.

**sio\_\$kill**

The value of this option is the kill character. It takes a character value, and the default is CTRL/X.

**sio\_\$line**

This value of this option is the SIO line number, an integer from 0 to **sio\_\$max\_line**. The value of this option cannot be changed with **sio\_\$control**.

**sio\_\$nlc\_delay**

The value of this option is the time delay in milliseconds following transmission of a line feed character, to allow for carriage motion or scrolling time, and so on. The default is 0.

**sio\_\$no\_NL**

This option is a Boolean value that specifies whether a newline character is transmitted as a carriage-return line-feed sequence or as a raw newline character. If **true**, newlines are transmitted as is. If **false**, newlines are transmitted as carriage-return line-feed sequences. The default is **false**.

**sio\_\$no\_echo**

This option is a Boolean value that specifies whether input characters are echoed back in the output. If **true**, input characters are not echoed. If **false**, input characters are echoed back to the sender. The default is **false**.

**sio\_\$parity**

The value of this option is the parity used by the SIO line. It can take any one of the following values:

**sio\_\$seven\_parity**

Even parity.

**sio\_\$no\_parity**  
No parity.

**sio\_\$odd\_parity**  
Odd parity.

If the parity is even or odd, then one bit is added to each character to enforce the chosen parity. If there are fewer than eight bits per character, the parity bit is delivered with the data in raw mode, but is stripped in cooked mode.

**sio\_\$quit\_enable**

This option is a Boolean value that specifies whether the node will respond to a quit character or break condition by interrupting the calling process. If **true**, a quit or break generates a quit fault. If **false**, a quit or break is ignored. The default is **false**. (In raw mode, break and quit are the only ways to generate a quit fault.)

**sio\_\$quitchr**

The value of this option is the quit character. It takes a character value, and the default is CTRL/].

**sio\_\$raw**

This option is a Boolean value that determines whether the line is in raw or cooked mode. If **true**, the line is in raw mode and each read returns all bytes received since the last call without any interpretation. If **false**, the line is in cooked mode and control codes are interpreted. The default value is **false**.

The input buffer is flushed whenever **sio\_\$raw** changes value.

**sio\_\$raw\_nl**

This option is a Boolean value that specifies whether a newline character is transmitted as a carriage-return line-feed sequence or as a raw newline character when the line is in raw mode. If **true**, newlines are transmitted as is while in raw mode. If **false**, newlines are transmitted as carriage-return line-feed sequences while in raw mode. The default is **false**.

**sio\_\$rts** This option is a Boolean value that determines the state of the Request To Send (RTS) line. The default is **true**.

**sio\_\$rts\_enable**

This option is a Boolean value that enables or disables the RTS line. If **true**, the RTS line is used with the CTS line for data flow control. For flow control to work, the CTS line must also be enabled. If **false**, no RTS/CTS flow control is provided. The default is **false**.

**sio\_\$send\_break**

The value of this option is the duration of a break condition on the SIO line, in milliseconds. A reasonable value for the **sio\_\$send\_break**

option is 200. A value other than 0 causes a break condition.

**sio\_\$speed**

The value of this option is the baud rate of the SIO line. It can have any of the following values: `sio_$50`, `sio_$75`, `sio_$110`, `sio_$134`, `sio_$150`, `sio_$300`, `sio_$600`, `sio_$1200`, `sio_$2000`, `sio_$2400`, `sio_$3600`, `sio_$4800`, `sio_$7200`, `sio_$9600`, or `sio_$19200`. The default is `sio_$9600`.

**sio\_\$speed\_force**

The value of this option is the baud rate of a partnered SIO line. The baud rate is forced to the value of this option, even if it is incompatible with the rate of the line's partner. If the value of `sio_$speed_force` is incompatible with the speed of the partnered line, the partnered line is forced to the default speed of `sio_$9600`. This option can have any of the following values: `sio_$50`, `sio_$75`, `sio_$110`, `sio_$134`, `sio_$150`, `sio_$300`, `sio_$600`, `sio_$1200`, `sio_$2000`, `sio_$2400`, `sio_$3600`, `sio_$4800`, `sio_$7200`, `sio_$9600`, or `sio_$19200`.

**sio\_\$stop\_bits**

The value of this option is the number of stop bits used by the SIO line. It can have any of the following values: `sio_$stop_1`, `sio_$stop_1_point_5`, or `sio_$stop_2`. The default is `sio_$stop_1`.

**sio\_\$susp\_enable**

This option is a Boolean value that enables or disables suspend signals from the calling process. If true, the SIO line honors suspend signals (faults). If false, the SIO line ignores suspend signals (faults).

**sio\_\$suspchr**

The value of this option is the suspend character. It takes a character value, and the default is CTRL/P.

**option\_value**

The value for the option specified by `sio_option`.

**status** The completion status. Possible values are

**sio\_\$bad\_option**

The option passed is not a valid SIO option.

**sio\_\$illegal\_strid**

The stream ID passed in the `sio_$` call is not open on an SIO line.

status

---

**Contents**

intro ..... STATUS-1

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every receipt, invoice, and bill should be properly filed and indexed for easy retrieval. This is particularly crucial for businesses that deal with a large volume of transactions, as it helps in identifying discrepancies and ensuring compliance with tax regulations.

Next, the document addresses the issue of budgeting and financial forecasting. It suggests that businesses should create a detailed budget at the beginning of each fiscal year, taking into account all expected income and expenses. Regularly comparing actual performance against the budget allows management to identify areas where costs are exceeding expectations and take corrective action.

The third section focuses on the importance of regular financial reviews. It recommends that businesses conduct monthly or quarterly reviews of their financial statements, including the balance sheet, income statement, and cash flow statement. These reviews provide valuable insights into the company's financial health and help in making informed decisions about future investments and operations.

Finally, the document discusses the role of technology in modern financial management. It highlights how accounting software and digital tools can streamline processes, reduce errors, and provide real-time access to financial data. However, it also cautions against over-reliance on technology, emphasizing the need for proper data security and backup procedures.

**NAME**

intro – status reporting types and constants

**SYNOPSIS (C)**

```
#include <apollo/base.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
```

**DESCRIPTION**

Most system calls supply their completion status in `status_$t` format. The `status_$t` type and the constant `status_$ok` are defined to help evaluate the status information supplied by system calls.

**Constants**

`status_$ok`

A constant used to check status. If the `all` instance of a completion status is equal to `status_$ok`, then the system call that supplied it was successful.

**Data Types**

`status_$t`

This is a variant record type with two instances. One instance has only one member, named `all`. The other instance has four members, named `fail`, `code`, `mode`, and `subsys`. Variables of the `status_$t` type require four bytes of storage.

**all** All 32 bits in the status code. If `all` is equal to `status_$ok`, the system call that supplied the status was successful.

**fail** The field labeled “f” in the diagram is the fail bit. If `fail` is set, the error was not within the scope of the module invoked, but occurred within a lower-level module.

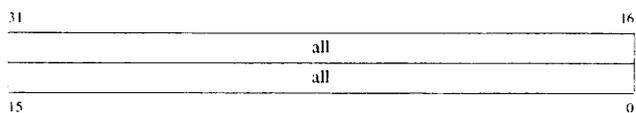
**subsys** This indicates the subsystem that encountered the error.

**mode** This indicates the module that encountered the error.

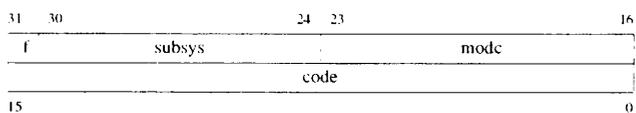
**code** This is a signed number that identifies the type of error that occurred.

STATUS\_\$INTRO

STATUS\_\$INTRO



OR



STATUS-2

Domain/OS Call Reference

# task

---

## Contents

intro .....	TASK-1
task_\$blast .....	TASK-9
task_\$create .....	TASK-10
task_\$exit .....	TASK-13
task_\$get_ec .....	TASK-14
task_\$get_handle .....	TASK-15
task_\$get_info .....	TASK-16
task_\$release .....	TASK-17
task_\$set_name .....	TASK-18
task_\$set_result .....	TASK-19
task_\$signal .....	TASK-20
task_tasking_\$enabled .....	TASK-21
task_\$yield .....	TASK-22

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, accounts payable, and accounts receivable. It also outlines the procedures for recording these transactions, including the use of double-entry bookkeeping to ensure that the books are balanced.

The second part of the document focuses on the analysis of the financial data. It explains how to interpret the various financial statements, such as the balance sheet, income statement, and cash flow statement. It provides a step-by-step guide to calculating key financial ratios and metrics, such as the current ratio, debt-to-equity ratio, and return on investment. The document also discusses the importance of comparing the company's performance to industry benchmarks and identifying areas for improvement.

The final part of the document covers the preparation of financial reports for management and external stakeholders. It provides a template for the financial statements and explains how to present the data in a clear and concise manner. It also discusses the importance of providing a narrative explanation of the results and highlighting any significant trends or risks. The document concludes with a summary of the key points and a final note on the importance of regular financial review and reporting.

**NAME**

intro – the Domain/OS task library

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>
```

**SYNOPSIS (Pascal)**

```
%include "/sys/ins/base.ins.pas"
%include "/sys/ins/task.ins.pas"
```

**DESCRIPTION**

The task\_\$ calls create and manage a multitasking environment within a single process. In a multitasking environment, multiple threads of execution (tasks) run within a single address space. Because task creation and switching are less expensive than process creation and switching, tasking is a useful way of breaking down any complex operation into separate pieces that run concurrently. It is particularly useful in Remote Procedure Call (RPC) servers that manage multiple remote requests in parallel.

**Creating a Task**

A process creates a task by calling task\_\$create, which returns a task handle to identify the task. You can obtain the handle of the currently running task by calling task\_\$get\_handle. (You can also optionally assign to the task an ASCII-string task name, up to 32 characters, by calling task\_\$set\_name.)

The first task in a process, the one that first calls task\_\$create, is known as the Distinguished Task (DT). After the first call to task\_\$create, tasking is enabled in the process, and any task can create a new task.

**Time Slicing and Priority Levels**

Tasks are able to run concurrently because of a scheduling scheme based on time slicing and priority levels. An initial priority level (from 1 to 9, where 9 is the highest) is assigned to a task when it is created. The Task Scheduler uses the priority level to determine which task (of the tasks that are ready) to run when one of the following conditions occurs:

- A time-slice interval ends.
- A task voluntarily yields the processor by calling task\_\$yield.
- A task blocks in user space by calling one of the following: ec2\_\$wait, ec2\_\$wait\_svc, time\_\$wait, msg\_\$wait, or mutex\_\$lock. (When a task blocks in the operating system the entire process is blocked, and another task is not started until the process returns to user space.)

As a task runs without blocking, its priority level is gradually lowered until it reaches the lowest priority level. It then runs at the lowest priority level until it blocks. Each time a task wakes after being blocked, the Task Scheduler reassigns the initial priority level to the task. The Task Scheduler algorithm favors tasks that run a short time between blocking, but ensures that all tasks eventually run: no task can lock out other tasks forever.

A task exists until one of the following events occurs:

- The task routine returns.
- The task routine calls `task_$exit`.
- An unhandled fault occurs.
- The task's lifetime, as defined by `task_$create`, expires.

#### Completion Eventcount

If a task is created with the `task_$intend_to_wait` option, the system maintains an eventcount that advances when the task completes. A task can wait for the completion of another by calling `task_$get_ec` to get the eventcount, and then using `ec2_$wait` to wait for it to advance. If a task is created with a completion eventcount, then some other task must call `task_$release` to release the task after it completes. The `task_$release` call will also supply the completion status and output value of the completed task to the caller.

#### Fault Inhibiting

Inhibiting faults in a tasking process inhibits all asynchronous signals, including those used for task switching. To inhibit all signals, including those that cause task switching, use `pfm_$inhibit` and `pfm_$enable`. For example, a tasking program could use `pfm_$inhibit` and `pfm_$enable` pairs to prevent other tasks from running during critical sections of code.

Fault inhibiting is always per task: one task can inhibit faults while another doesn't. If a fault is pending when a non-inhibited task begins to run, the fault is signaled then. Also, `pfm_$inhibit` does not prevent a task switch from occurring when a task blocks or voluntarily yields the processor.

#### Fault Handlers

Because fault handlers are established on a per-process basis, the same fault handler executes regardless of the task that established it or the task that is running when the fault occurs. A handled fault is not propagated to the DT.

#### Clean-Up Handlers

Clean-up handlers are established per task; that is, a clean-up handler is invoked only in the task that established it. A synchronous fault is delivered to the clean-up handler in the task in which the synchronous fault occurs. An asynchronous fault is delivered to the DT the next time it runs. The Task Manager sets up a default clean-up handler at the base of a task's stack when the task is created.

A task can exit with cleanup either by returning or by calling `task_$exit`. A task can also cause another task to exit cleanly by calling `task_$signal`. As a last resort, a task can destroy another task without cleanup by calling `task_$blast`.

#### Programming Restrictions

Because tasks are time-sliced, an application using tasking must ensure that all code it uses is "re-entrant"; that is, designed so that multiple tasks can execute it concurrently.

For an example of code that is not re-entrant, suppose an application (written in C) uses two tasks, T1 and T2. Each task calls a function `add_to_table` that adds an integer to a table (array) of integers. The procedure uses two global variables, `num_of_entries` and `table`.

```
int num_of_entries = 0;
int table[100];

void
add_to_table(int x)
{
    table[num_of_entries] = x;
    num_of_entries = num_of_entries + 1;
}
```

Task T1 assigns `x` to `table[num_of_entries]` but is then suspended. Meanwhile, task T2 starts running and calls `add_to_table`. Since task T1 has not yet updated `num_of_entries`, task T2 overwrites the entry that task T1 has made. Presumably, the application intends that the table and entry count should reflect the calls made by both tasks; consequently, `add_to_table` is not re-entrant.

It is often possible to safely use code that is not re-entrant. For example, if a system library that is not re-entrant is called solely from one task, then it does not matter that the library is not re-entrant. Also, some libraries return handles to static data structures, where the handle is the only pathway to that storage. Consequently, if only one task ever uses a particular handle when it makes calls that are not re-entrant, no problems will occur. For example, `ios_$open` is not re-entrant, but a single task may safely use it.

If you create a library and you intend it to run with tasking applications, you should make the manager re-entrant by using Mutual Exclusion (mutex) locks. Mutex locks ensure that only one task at a time has access to the manager's data structures. In order to use a data structure, the task must first request and receive the lock associated with it. For example, to make the `add_to_table` procedure re-entrant, add mutex locks to it as follows:

```
int num_of_entries = 0;
int table[100];
mutex_$lock_rec_t table_lock;

void
add_to_table(int x)
{
    mutex_$lock(table_lock, mutex_$wait_forever);
    table[num_of_entries] = x;
    num_of_entries = num_of_entries + 1;
    mutex_$unlock(table_lock);
}
```

Note that the application must call `mutex_$init` before it makes the first call to the `add_to_table` procedure.

Not all Domain/OS libraries are re-entrant. The following table categorizes Domain/OS calls into those that are probably re-entrant (that is, they are most likely re-entrant but have not been thoroughly tested for this feature), partially re-entrant (they return handles as described above), not re-entrant, and those that are definitely re-entrant.

Re-entrant Properties of Domain/OS Calls			
Probably Re-entrant	Partially Re-entrant	Not Re-entrant	Re-entrant
ec2	cal	error	ctm
ms	fpp	gmf	ev
mutex	pbu	ios	gmr
name	proc2	ipc	gmr3d
pfm	sio	pad	gpr
procl	smd	stream	mbx
rws	time	vfmt	pbufs
sfcf	tpad		pgm
task	vec		prf
			trait

Domain/OS calls without side effects (`time_$clock` for example) are generally re-entrant. Calls that open, create, or close objects (`ios_$open` and `ipc_$delete` for example) are not re-entrant.

The following additional restrictions apply to tasking:

- UNIX system and library calls are not supported. They may work, but they are not guaranteed to.
- Asynchronous fault handlers must not make any `task_$` calls.
- `Ec2_$wait_svc` behaves exactly like `ec2_$wait` when tasking is enabled. The reason is that the Task Manager does not know which task to notify when an asynchronous fault handler has returned to the point at which the fault occurred.

#### Constants

`task_$max_priority`  
The highest task priority.

`task_$name_max_len`  
The maximum number of bytes in a task name.

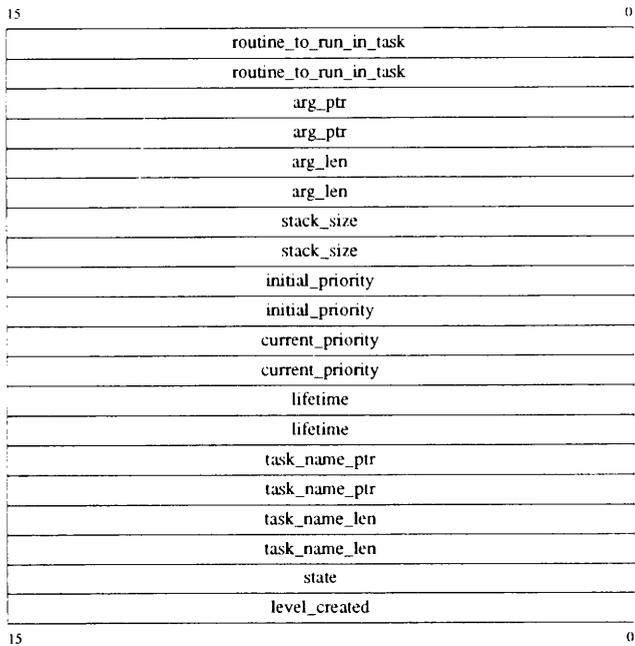
`task_$dt_handle`  
The task handle for the Distinguished Task (DT).

#### Data Types

**task\_\$handle\_t**  
A task handle.

**task\_\$info\_ptr**  
A pointer to type task\_\$info\_t.

**task\_\$info\_t**  
A record type for passing task information. The diagram below illustrates the task\_\$info\_t data type.



**routine\_to\_run\_in\_task**  
A pointer to the task routine.

**arg\_ptr** Pointer to the task's argument vector.

**arg\_len** The number of bytes in the task's argument vector.

**stack\_size**

The number of bytes in the task's stack.

**initial\_priority**

The initial priority of the task.

**current\_priority**

The priority at which the task is currently running.

**lifetime** The lifetime of the task. It is a value of type `task_$lifetime_t`.

**task\_name\_ptr**

A pointer to the name of the task.

**task\_name\_len**

The number of bytes in the task name.

**state** The run state of the task. It is a value of type `task_$run_state_t`.

**level\_created**

The program level at which the task was created.

**task\_\$lifetime\_t**

An enumerated type for specifying the lifetime of a task. (In Pascal, the values of type `task_$lifetime_t` are constants.) It takes one of the following values:

**task\_\$forever**

Create a task that lives independent of the process that created it.

**task\_\$until\_level\_exit**

The task lives until the level at which it was created is released.

**task\_\$until\_exec\_or\_level\_exit**

The task lives until the level at which it was created is released or overlaid by a UNIX `exec(2)` call.

**task\_\$name\_ptr**

A pointer to a task name.

**task\_\$option\_set\_t**

A small set type for specifying options to `task_$create`. There is currently only one value:

**task\_\$intend\_to\_wait**

Create a task completion eventcount that advances when the new task terminates. When created with the `task_$intend_to_wait` option, a task is suspended when it completes pending a `task_$release` call.

When created without the `task_$intend_to_wait` option, a task is not suspended when it completes; instead it terminates immediately on completion, and the completion status and output value of the task will not be accessible.

**task\_\$routine\_pt**

A pointer to a task routine. A task routine must not return a value and must take two arguments: a pointer to an argument vector, followed by the number of bytes in the argument vector. The argument vector may contain anything.

In C, a valid task routine declaration is

```
void task_routine(
    void *arg_pointer,
    int arg_length)
```

In Pascal, a valid task routine declaration is

```
procedure task_routine(
    in arg_ptr: univ_ptr;
    in arg_len: integer32);
options(val_param);
```

**task\_\$run\_state\_t**

An enumerated type for describing the run state of a task. It takes one of the following values:

**task\_\$ready**

The task is ready to run.

**task\_\$waiting**

The task is waiting.

**Errors****task\_\$bad\_sec\_wait**

Internal error: bad status from ec2\_\$wait.

**task\_\$bad\_handle**

Invalid task handle.

**task\_\$cant\_blast\_dt**

Attempted to blast the distinguished task.

**task\_\$cant\_blast\_yourself**

Task attempted to blast itself.

**task\_\$exit\_fault**

Task exited.

**task\_\$invalid\_lifetime**

Invalid task lifetime.

**task\_\$invalid\_name**

Task name is too long.

**task\_\$invalid\_priority**

Invalid task priority.

**task\_\$lib\_init\_failed**

Internal error: library initialization failed.

- task\_\$no\_backgrnd\_from\_foregrnd**  
A foreground task attempted to create a background task.
- task\_\$no\_completion\_ec**  
Task does not have a completion eventcount.
- task\_\$no\_room**  
Not enough virtual memory left to create task.
- task\_\$not\_completed**  
Task not yet completed.
- task\_\$not\_found**  
Task does not exist.
- task\_\$not\_init**  
Task Manager has not been initialized.
- task\_\$stack\_corrupted**  
A task stack was corrupted.
- task\_\$stack\_overflow**  
A task stack overflowed.
- task\_\$too\_many\_tasks**  
Attempted to exceed the maximum allowable number of tasks in a process.

**SEE ALSO**

ec2\_\$intro, fault\_\$intro, mutex\_\$intro, pfm\_\$intro.

**NAME**

`task_$blast` – kill a task without cleanup

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
void task_$blast(
    task_$handle_t task_handle,
    status_$t *status)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
procedure task_$blast(
    in task_handle: task_$handle_t;
    out status: status_$t);
```

**DESCRIPTION**

`Task_$blast` kills the task identified by *task\_handle*. Because the task is immediately destroyed without its clean-up handlers being invoked, `task_$blast` should be used only as a last resort.

*task\_handle*

The handle of the task.

*status*

The completion status.

**NOTES**

`Task_$exit` and `task_$signal` can terminate a task via a signal.

**SEE ALSO**

`pfm_$intro`, `task_$create`, `task_$exit`, `task_$get_handle`, `task_$signal`.

## NAME

`task_create` – create a task

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
task_handle_t task_create(
    void (*routine_pointer)(),
    char *arg_pointer,
    long arg_length,
    long stack_size,
    long task_priority,
    task_lifetime_t life_time,
    task_option_set_t task_options,
    status_t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
function task_create(
    in routine_pointer: task_routine_p;
    in arg_pointer: univ_ptr;
    in arg_length: integer32;
    in stack_size: integer32;
    in task_priority: integer32;
    in life_time: task_lifetime_t;
    in task_options: task_option_set_t;
    out status: status_t): task_handle_t;
```

## DESCRIPTION

`Task_create` creates a new task and returns a handle for it. `Task_create` allocates `stack_size` bytes for the task's stack, then creates an initial context by calling the routine that `routine_pointer` points to.

*routine\_pointer*

A pointer to the routine to run in the task. The task routine must not return a value and must take two arguments: a pointer to an argument vector, followed by the number of bytes in the argument vector. The argument vector may contain anything.

In C, a valid task routine declaration is

```
void task_routine(
    void *arg_pointer,
    int arg_length)
```

In Pascal, a valid task routine declaration is

```
procedure task_routine(
  in arg_ptr: univ_ptr;
  in arg_len: integer32);
  options(val_param);
```

*arg\_pointer*

A pointer to the argument vector for the routine at *routine\_pointer*. The contents and format of the argument vector is not defined by the task library. If *arg\_length* is 0, *arg\_pointer* is passed directly to the routine at *routine\_pointer*; otherwise, the argument vector is copied from *arg\_pointer* onto the stack for the created task, and the task gets a pointer to that copy in its first argument.

*arg\_length*

The number of bytes at *arg\_pointer* to pass in the argument vector. The value of *arg\_length* is passed to the routine at *routine\_pointer* as its second argument when it is called.

*stack\_size*

The number of bytes to allocate for the new task's stack. In general, *stack\_size* should be at least 64K bytes so it can accommodate system calls, some of which require a large stack.

Task\_create increases the size of the stack to hold the argument vector, so *stack\_size* should not include the size of the argument vector.

*task\_priority*

The initial priority of the task. The priority levels are from 1 to 9, where 9 is the highest.

*life\_time*

The lifetime of the new task. Specify one of the following values:

**task\_forever**

Create a task that lives independent of the process that created it.

**task\_until\_level\_exit**

Create a task that lives until the level at which it was created is released.

**task\_until\_exec\_or\_level\_exit**

Create a task that lives until the level at which it was created is released or overlaid by a UNIX exec(2) call.

The system does not ensure that a task's code is available for the duration of the task's lifetime; that is left up to the application. Thus, if *life\_time* is **task\_forever**, the task's code must exist in a global library.

*task\_options*

A small set of options when creating a new task. Currently there is one option:

**task\_intend\_to\_wait**

Create a task completion eventcount that advances when the created task terminates. When created with the **task\_intend\_to\_wait** option, a task is suspended when it completes pending a **task\_release** call.

If **task\_intend\_to\_wait** is not specified in *task\_options* — that is, if *task\_options* is 0 (or [] in Pascal) — then **task\_create** does not create a completion eventcount. When created without the **task\_intend\_to\_wait** option, a task is not suspended when it completes; instead it terminates immediately on completion, and the completion status and output value of the task will not be accessible.

*status* The completion status.

**SEE ALSO**

**task\_get\_ec**, **task\_release**, **task\_set\_result**.

**NAME**

**task\_\$exit** – exit a task

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
void task_$exit();
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
procedure task_$exit;
```

**DESCRIPTION**

Task\_\$exit terminates the calling task by signaling a `task_$exit_fault` fault. Consequently, any clean-up handlers for that fault established by the task will run.

**SEE ALSO**

`pfm_$intro`, `task_$blast`, `task_$create`, `task_$get_handle`, `task_$signal`.

## NAME

`task_$get_ec` – get a completion eventcount

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
void task_$get_ec(
    task_$handle_t task_handle,
    ec2_$ptr_t *eventcount_pointer,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
procedure task_$get_ec(
    in task_handle: task_$handle_t;
    out eventcount_pointer: ec2_$ptr_t;
    out status: status_$t);
```

## DESCRIPTION

`Task_$get_ec` returns a pointer to the completion eventcount for the task identified by *task\_handle*. The task must have been created with a completion eventcount; that is, the `task_$create` call that created it must have specified the `task_$intend_to_wait` option.

*task\_handle*

The handle of the task, in `task_$handle_t` format.

*eventcount\_pointer*

A pointer to an eventcount.

*status* The completion status.

## SEE ALSO

`ec2_$intro`, `task_$get_handle`, `task_$release`, `task_$set_result`.

TASK\_\$GET\_HANDLE

TASK\_\$GET\_HANDLE

**NAME**

`task_$get_handle` – return the task handle

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
task_$handle_t task_$get_handle();
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
function task_$get_handle: task_$handle_t;
```

**DESCRIPTION**

`Task_$get_handle` returns the task handle of the calling task.

**SEE ALSO**

`task_$create`.

## NAME

`task_$get_info` – get information about a task

## SYNOPSIS (C)

```
#include <apollo/base.h>
```

```
#include <apollo/task.h>
```

```
void task_$get_info(
    task_$handle_t task_handle,
    task_$info_pt info_pointer,
    task_$handle_t *next_handle,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
```

```
%include '/sys/ins/task.ins.pas';
```

```
procedure task_$get_info(
    in task_handle: task_$handle_t;
    in info_pointer: task_$info_pt;
    out next_handle: task_$handle_t;
    out status: status_$t);
```

## DESCRIPTION

`Task_$get_info` gets information about the task identified by `task_handle`.

*task\_handle*

A task handle.

*info\_pointer*

The task information.

*next\_handle*

The handle of the next task for which information can be obtained. The handles for tasks in a process are in a circular list. Repeated calls to `task_$get_info` with `task_handle` set to the `next_handle` supplied by the previous call will eventually return a `next_handle` equivalent to the `task_handle` when `task_$get_info` was first called, indicating that all tasks have been described.

*status* The completion status.

## SEE ALSO

`task_$create`, `task_$get_handle`.

**NAME**

**task\_\$release** – release a task and report

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>

void task_$release(
    task_$handle_t task_handle,
    status_$t *completion_status,
    long *output_value,
    status_$t *status)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';

procedure task_$release(
    in task_handle: task_$handle_t;
    out completion_status: status_$t;
    out output_value: integer32;
    out status: status_$t);
```

**DESCRIPTION**

**Task\_\$release** returns the completion status and output value of the task identified by *task\_handle*, and then terminates the task. It may be called only if the task was created with a completion eventcount.

*task\_handle*

A task handle.

*completion\_status*

The completion status of the task identified by *task\_handle*. If the task completed because it received a fault signal it couldn't handle, *completion\_status* will be the fault code received. Otherwise, *completion\_status* will be *status\_\$ok* unless the task changed it by calling *task\_\$set\_result*.

*output\_value*

The output value of the task identified by *task\_handle*. The task's output value will be 0 unless the task changed it by calling *task\_\$set\_result*.

*status*

The completion status. If *task\_\$release* is called before the task has completed, it returns with a completion status of *task\_\$not\_completed*, and *completion\_status* and *output\_value* are undefined. In general, *task\_\$release* should not be called until the completion eventcount of the task specified by *task\_handle* is advanced.

**SEE ALSO**

*task\_\$create*, *task\_\$exit*, *task\_\$get\_ec*, *task\_\$set\_result*.

**NAME**

`task_$set_name` – name a task

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
void task_$set_name(
    task_$handle_t task_handle,
    task_$name_pt name_pointer,
    long name_length,
    status_$t *status)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
procedure task_$set_name(
    in task_handle: task_$handle_t;
    in name_pointer: task_$name_pt;
    in name_length: integer32;
    out status: status_$0);
```

**DESCRIPTION**

`Task_$set_name` associates the name pointed to by `name_pointer` with the task identified by `task_handle`. The name is copied into the task's information record, and is accessible via `task_$get_info`.

*task\_handle*

A task handle.

*name\_pointer*

A pointer to the name to be associated with the task.

*name\_length*

The number of significant bytes at `name_pointer`. `Name_length` should be `task_$name_max_len` or less.

*status*

The completion status.

**SEE ALSO**

`task_$create`, `task_$get_handle`.

## NAME

`task_$set_result` – change the completion status and output value

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/task.h>

void task_$set_result(
    status_$(completion_status),
    long output_value,
    status_$( *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';

procedure task_$set_result(
    in completion_status: status_$(t);
    in output_value: integer32;
    out status: status_$(t);
```

## DESCRIPTION

`Task_$set_result` sets the completion status and output value of the calling process to *completion\_status* and *output\_value*, respectively. It may be called only by a task created with a completion eventcount.

*completion\_status*

The completion status for the calling task.

*output\_value*

The output value for the calling task.

*status* The completion status of `task_$set_result`.

## NOTES

The completion status and output value of a task are reported by `task_$release` when the task is terminated.

## SEE ALSO

`task_$create`, `task_$get_handle`, `task_$release`.

## NAME

`task_$signal` – signal a task

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/task.h>

void task_$signal(
    task_$handle_t task_handle,
    status_$t fault_status,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';

procedure task_$signal(
    in task_handle: task_$handle_t;
    in fault_status: status_$t;
    out status: status_$t);
```

## DESCRIPTION

`Task_$signal` sends the signal specified by `fault_status` to the task specified by `task_handle`. `Task_$signal` will activate any fault or clean-up handlers established by the task on `task_handle` to handle the fault specified by `fault_status`. If the task is not running or has inhibited faults, it will be signaled the next time it is running with faults enabled.

*task\_handle*

A task handle.

*fault\_status*

The signal to send to the task on `task_handle`. Also, the fault code passed to any fault or clean-up handler established to handle the fault.

*status*

The completion status of `task_$signal`.

## SEE ALSO

`pfm_$intro`, `task_$blast`, `task_$create`, `task_$exit`, `task_$get_handle`.

TASK\_TASKING\_\$ENABLED

TASK\_TASKING\_\$ENABLED

**NAME**

`task_$tasking_enabled` – determine whether tasking is enabled

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/task.h>
```

```
boolean task_$tasking_enabled(void)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/task.ins.pas';
```

```
function task_$tasking_enabled: boolean;
```

**DESCRIPTION**

`Task_$tasking_enabled` returns `true` if tasking is enabled, or `false` if tasking is not enabled. Tasking is enabled when the Distinguished Task (DT) first makes a call to `task_$create`.

**NAME**

`task_yield` – yield the processor

**SYNOPSIS (C)**

```
#include <apollo/base.h>
```

```
#include <apollo/task.h>
```

```
void task_yield(void);
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
```

```
%include '/sys/ins/task.ins.pas';
```

```
procedure task_yield;
```

**DESCRIPTION**

`Task_yield` yields the processor to the next ready task at the same priority level as the calling task. If there are no other ready tasks at that level, the calling task continues to run.

**SEE ALSO**

`task_exit`.

# time

---

## Contents

intro .....	TIME-1
time_\$clock .....	TIME-3
time_\$get_ec .....	TIME-4
time_\$wait .....	TIME-6

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, accounts payable, and accounts receivable. It also outlines the procedures for recording these transactions, including the use of double-entry bookkeeping to ensure that the books are balanced.

The second part of the document focuses on the analysis of the recorded data. It explains how to calculate key financial ratios and metrics, such as the gross profit margin, net profit margin, and return on investment. These calculations are essential for understanding the overall performance of the business and identifying areas for improvement. The document also discusses the importance of comparing the current period's performance with that of previous periods and with industry benchmarks to provide context for the results.

The final part of the document addresses the reporting requirements for the financial data. It outlines the format and content of the financial statements, including the balance sheet, income statement, and cash flow statement. It also discusses the importance of providing clear and concise explanations for any significant changes or trends in the data. The document concludes by emphasizing the need for transparency and accuracy in all financial reporting to build trust with stakeholders and ensure compliance with regulatory requirements.

**NAME**

intro – the Domain/OS time service

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/time.h>
```

**SYNOPSIS (Pascal)**

```
%include 'sys/ins/base.ins.pas';
%include 'sys/ins/time.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include 'sys/ins/base.ins.ftn'
%include 'sys/ins/time.ins.ftn'
```

**DESCRIPTION**

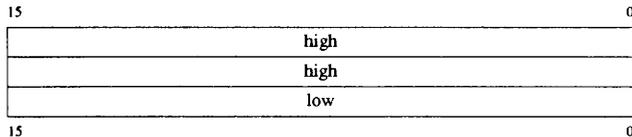
Domain/OS represents time as the number of 4-microsecond intervals that have elapsed since midnight (00:00) on 1 January 1980 (UTC). (“UTC” is an abbreviation for “Coordinated Universal Time,” which is just another, less parochial, term for Greenwich Mean Time or “GMT.”) The count of elapsed 4-microsecond intervals is a 48-bit integer value of type `time_$clock_t`. An integer of type `time_$clock_t` is called a “clock value.”

The Domain/OS time service reports system time, provides an eventcount that is advanced at regular intervals, and allows the timed suspension of processes. The `time_$` interface consists of the following calls:

<code>time_\$clock</code>	get the system clock value
<code>time_\$get_ec</code>	get a pointer to the time eventcount
<code>time_\$wait</code>	wait for an interval

**Data Types****time\_\$clock\_t**

A record type for system clock values. A value of type `time_$clock_t` is a 48-bit integer count of 4-microsecond clock periods. A clock value has the following format:



**high** The most significant 32 bits of the clock value.

**low** The least significant 16 bits of the clock value.

**time\_\$\$clockh\_t**

A 32-bit unsigned integer for holding the most significant bits of a clock value.

**time\_\$\$key\_t**

An enumerated type for choosing a time eventcount. It currently can take on only one value:

**time\_\$\$clockh\_key**

An eventcount that is advanced about every 1/4 second.

**time\_\$\$rel\_abs\_t**

An enumerated type for specifying the type of clock value passed in an argument. It can take one of the following values:

**time\_\$\$relative**

The clock value specifies an interval.

**time\_\$\$absolute**

The clock value specifies a UTC time.

#### Errors

**time\_\$\$bad\_key**

Bad key to `time_$$get_ec`.

**time\_\$\$wait\_quit**

Wait interrupted by quit fault.

## NAME

`time_$clock` – get the system clock value

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/time.h>
```

```
void time_$clock(
    time_$clock_t *clock_value)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/time.ins.pas';
```

```
procedure time_$clock(
    out clock_value: time_$clock_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/time.ins.ftn'
```

```
integer*2 clock_value(3), clockh
integer*4 clockl
```

```
equivalence (clockl, clock_value(2)), (clockh, clock_value(1))
```

```
call time_$clock(clock_value)
```

## DESCRIPTION

`Time_$clock` supplies the system clock value in `clock_value`.

`clock_value`

The value of the system clock.

## SEE ALSO

`cal_$apply_local_offset`, `cal_$get_local_time`.

## NAME

`time_$get_ec` – get a pointer to the time eventcount

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/time.h>
```

```
void time_$get_ec(
    time_$key_t &time_key,
    ec2_$ptr_t *eventcount_pointer,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/time.ins.pas';
```

```
procedure time_$get_ec(
    in time_key: time_$key_t;
    out eventcount_pointer: ec2_$ptr_t;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/time.ins.ftn'
```

```
integer*4 status, ec_value
integer*2 time_key, event(3)
```

equivalence (ec\_value, event(1))

```
integer*4 eventcount_pointer
pointer /eventcount_pointer/ event
```

```
call time_$get_ec(time_key, eventcount_pointer, status)
```

## DESCRIPTION

`Time_$get_ec` supplies a pointer to an eventcount that is advanced about every 1/4 second. The incrementing interval is nominally 262,144 microseconds, but the exact interval varies with the system load.

*time\_key*

A key specifying which time eventcount the system should return. The only defined value currently is `time_$clockh_key`, because there is only one time eventcount.

*eventcount\_pointer*

A pointer to the eventcount specified by *time\_key*.

TIME\_\$GET\_EC

TIME\_\$GET\_EC

*status* The completion status.

SEE ALSO

ec2\_\$intro.

## NAME

`time_$wait` – wait for an interval

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/time.h>

void time_$wait(
    time_$rel_abs_t &rel_abs,
    time_$clock_t &clock_value,
    status_$t *status)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/time.ins.pas';

procedure time_$wait(
    var rel_abs: time_$rel_abs_t ;
    in clock_value: time_$clock_t ;
    out status: status_$t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/time.ins.ftn'

integer*2 clock_value(3), clockh, rel_abs
integer*4 clockl, status

equivalence (clockl, clock_value(2)), (clockh, clock_value(1))

call time_$wait(rel_abs, clock_value, status)
```

## DESCRIPTION

`Time_$wait` returns after the interval specified by `rel_abs` and `clock_value`.

`rel_abs` The type of clock value supplied supplied in `clock_value`. It can have one of the following values:

`time_$relative`

`Clock_value` specifies a relative interval to wait before returning. `Time_$wait` will return after the number of 4-microsecond periods specified by `clock_value` have elapsed.

`time_$absolute`

`Clock_value` specifies a system clock value at which `time_$wait` should return.

`clock_value`

A clock value specifying a relative or absolute time to wait for before returning.

TIME\_WAIT

TIME\_WAIT

*status* The completion status. If the completion status is `time_wait_quit`, then `time_wait` returned because of an asynchronous fault, not because the specified interval elapsed.

**NOTES**

Note that `time_wait` expects a Coordinated Universal Time (UTC) time in *clock\_value* if *rel\_abs* equals `time_absolute`. To convert local time into UTC time, use `cal_remove_local_offset`.

**SEE ALSO**

`ec2_wait`, `ec2_wait_slow_io`, `ec2_wait_svc`.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, customer orders, and supplier invoices. It also outlines the procedures for recording these transactions, including the use of specific forms and the assignment of responsibilities to different staff members.

The second part of the document focuses on the analysis of the recorded data. It describes various methods for identifying trends and anomalies in the financial performance. This includes comparing current data with historical trends, as well as benchmarking against industry standards. The document also discusses the importance of regular reviews and audits to ensure that the records are accurate and up-to-date. It provides a step-by-step guide for conducting these reviews, from the initial data collection to the final reporting and analysis.

The final part of the document discusses the implications of the financial data for the overall business strategy. It explains how the recorded information can be used to make informed decisions about resource allocation, pricing, and marketing. The document also highlights the importance of transparency and communication in the financial reporting process, ensuring that all stakeholders have access to the necessary information to make their own assessments.

tone

---

**Contents**

intro .....	TONE-1
tone_\$time .....	TONE-2

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial statements. This includes not only sales and purchases but also expenses, income, and any other financial activity.

The second part of the document provides a detailed breakdown of the accounting process. It starts with the identification of the accounting cycle, which consists of eight steps: identifying the accounting cycle, analyzing and journalizing the transactions, posting to the ledger, determining debits and credits, preparing a trial balance, adjusting the entries, preparing financial statements, and closing the books.

The third part of the document discusses the importance of the trial balance. It explains that the trial balance is a statement that lists all the accounts and their balances at a specific point in time. It is used to check the accuracy of the accounting records and to ensure that the debits equal the credits.

The fourth part of the document discusses the importance of the financial statements. It explains that the financial statements provide a summary of the company's financial performance over a period of time. The three main financial statements are the income statement, the balance sheet, and the cash flow statement.

The fifth part of the document discusses the importance of the closing process. It explains that the closing process involves transferring the balances of the temporary accounts (revenues, expenses, and dividends) to the permanent accounts (retained earnings and dividends). This process is necessary to prepare the books for the next accounting period.

**NAME**

intro – make a noise

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/tone.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tone.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tone.ins.ftn'
```

**DESCRIPTION**

This section describes the `tone_$time` call, which allows software to command the node it's running on to make a noise. Only Apollo workstations shipped after 19 April 1982 can make a noise on command from software.

**NAME**

**tone\_\$time** – make a noise of a specified duration

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/tone.h>
```

```
void tone_$time(time_$clock_t &time)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tone.ins.pas';
```

```
procedure tone_$time(in time: time_$clock_t);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tone.ins.ftn'
```

```
integer*4 clockl
integer*2 clockh, time(3)
```

equivalence (clockl, time(2)), (clockh, time(1))

```
call tone_$time(time)
```

**DESCRIPTION**

This call commands the node to make a noise for the length of time specified in *time*.

*time* This is the duration of the noise in **time\_\$clock\_t** format.

# tpad

---

## Contents

intro .....	TPAD-1
tpad_\$inq_dtype .....	TPAD-3
tpad_\$inquire .....	TPAD-4
tpad_\$re_range .....	TPAD-6
tpad_\$set_cursor .....	TPAD-7
tpad_\$set_mode .....	TPAD-9



**NAME**

intro – locator (touchpad) manager calls

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/tpad.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tpad.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tpad.ins.ftn'
```

**DESCRIPTION**

The `tpad_$` system calls manage locating devices.

**Data Types****tpad\_\$mode\_t**

This is an enumerated type for specifying how locator input is translated in to cursor movement. Variables of this type can take one of the following values:

**tpad\_\$absolute**

Absolute mode maps the pointing space to a part of the screen dictated by the scaling factor and the origin value. Thus, there is a one-to-one mapping between points in the pointing space and a subset of points on the screen.

**tpad\_\$relative**

In relative mode, the cursor responds only to device movement relative to the current position. The response of the cursor simulates response to mouse movement. This is the only meaningful mode to use with a mouse.

**tpad\_\$rel\_abs**

In relative/absolute mode, the cursor responds as in absolute mode on first contact to establish a new current position. After first contact, the cursor responds as in relative mode. Momentary lifting of the finger from a touchpad (for less than about half a second) does not affect the cursor position.

**smd\_\$pos\_t**

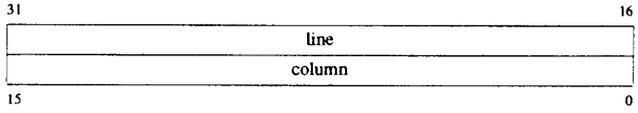
A record type for specifying display positions.

**line** The vertical coordinate.

**column** The horizontal coordinate.

TPAD\_\$INTRO

TPAD\_\$INTRO



TPAD-2

Domain/OS Call Reference

## NAME

**tpad\_\$inq\_dtype** – return the last locator used

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/tpad.h>
```

```
tpad_$dev_type_t tpad_$inq_dtype(void)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tpad.ins.pas';
```

```
function tpad_$inq_dtype : tpad_$dev_type_t;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tpad.ins.ftn'
```

```
integer*2 device
device = tpad_$inq_dtype
```

## DESCRIPTION

This call returns a value indicating the last locator used for input. The returned value can be one of the following:

**tpad\_\$have\_bitpad**

A bitpad provided the last locator input.

**tpad\_\$have\_mouse**

A mouse provided the last locator input.

**tpad\_\$have\_touchpad**

A touchpad provided the last locator input.

**tpad\_\$unknown**

No locator input was received since the node was last booted.

## SEE ALSO

tpad\_\$inquire.

**NAME**

**tpad\_\$inquire** – get information about the current locator response

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/tpad.h>

void tpad_$inquire(
    tpad_$mode_t *mode,
    short *x_scale,
    short *y_scale,
    short *hysteresis,
    smd_$pos_t *origin)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tpad.ins.pas';

procedure tpad_$inquire(
    out mode: tpad_$mode_t;
    out x_scale: integer;
    out y_scale: integer;
    out hysteresis: integer;
    out origin: smd_$pos_t);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tpad.ins.ftn'

integer*2 mode, x_scale, y_scale, hysteresis
integer*2 line, column, origin(2)

equivalence (line, origin(1)), (column, origin(2))

call tpad_$inquire(mode, x_scale, y_scale, hysteresis, origin)
```

**DESCRIPTION**

This call supplies information about the response of the current locator device. Use the information from this call to save the locator configuration for later restoration with the **tpad\_\$set\_mode** call, or to change one aspect of the locator response without changing any other aspects.

**mode** The cursor mode. It can be one of the following predefined values:

**tpad\_\$absolute**

Absolute mode maps the locator to a part of the screen dictated by the scaling factor and the origin value. Thus, there is a one-to-one mapping between points in the locator space and a subset of points on the

screen.

**tpad\_\$relative**

In relative mode, the cursor responds only to locator movement relative to the current position. The response of the cursor simulates response to mouse movement.

**tpad\_\$rel\_abs**

In relative/absolute mode, the cursor responds as in absolute mode on first contact to establish a new current position. After first contact, the cursor responds as in relative mode. Momentary lifting of the finger from a touchpad (for less than about half a second) does not affect the current cursor position.

*x\_scale* The scale factor in the x-dimension.

*y\_scale* The scale factor in the y-dimension.

***hysteresis***

The hysteresis factor, in pixels.

The hysteresis factor prevents minor, unintentional movements from changing the cursor location. It effectively defines a box around the current location. The cursor does not move while the current location stays within the hysteresis box.

If the current location moves beyond the box, the distance the cursor moves is the difference between the new location and the past location less the hysteresis factor. The default hysteresis factor is 5.

*origin* The scale origin for x- and y-dimensions in **smd\_\$pos\_t** format.

**SEE ALSO**

**tpad\_\$inq\_dtype**, **tpad\_\$re\_range**, **tpad\_\$set\_cursor**, **tpad\_\$set\_mode**.

## NAME

`tpad_$re_range` – re-establishes the touchpad raw data range

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/tpad.h>
```

```
void tpad_$re_range(void)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tpad.ins.pas';
```

```
procedure tpad_$re_range;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tpad.ins.ftn'
```

```
call tpad_$re_range
```

## DESCRIPTION

This call re-establishes the touchpad raw data range over the next 1000 data points. This is also done for you at system boot.

## NAME

`tpad_$set_cursor` – re-establish the locator origin in relative mode

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/tpad.h>
```

```
void tpad_$set_cursor(
    smd_$pos_t &origin)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tpad.ins.pas';
```

```
procedure tpad_$set_cursor(
    in origin: smd_$pos_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tpad.ins.ftn'
```

```
integer*2 line, column, origin(2)
```

```
equivalence (line, origin(1)), (column, origin(2))
```

```
call tpad_$set_cursor(origin)
```

## DESCRIPTION

This call is useful for relative and absolute/relative mode only and is primarily used with touchpad and bitpad locating devices.

The system remembers the last cursor position delivered by the locator. When new input comes from a locator in relative mode, a displacement is computed and applied to the previous locator position. The `tpad_$set_cursor` call makes the system forget the previous locator position, and use the value passed in the call instead. The next displacement will be computed from the position specified in the `tpad_$set_cursor` call instead of the locator's previous position. `tpad_$set_cursor` can be called at any time and affect all subsequent locator input.

When using a touchpad or bitpad in relative mode, the origin is automatically re-established when the user takes his finger from the touchpad for more than one eighth of a second. One effect of relative mode is that the cursor doesn't move the next time the user touches the pad unless `tpad_$set_cursor` was called in the interim.

When using a touchpad in absolute/relative mode, a call to `tpad_$set_cursor` is only useful while the locator is functioning in the relative phase; that is, after the first touch and before the operator lifts his finger for more than one half second.

TPAD\_\$SET\_CURSOR

TPAD\_\$SET\_CURSOR

*origin* The screen position that will be the origin for subsequent input from the the locator in `smd_$pos_t` format.

**SEE ALSO**

`tpad_$inquire`, `tpad_$set_mode`.

## NAME

`tpad_$set_mode` – set pointing device response characteristics

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/tpad.h>
```

```
void tpad_$set_mode(
    tpad_$mode_t &mode,
    short &x_scale,
    short &y_scale,
    short &hysteresis,
    smd_$pos_t &origin)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/tpad.ins.pas';
```

```
procedure tpad_$set_mode(
    in mode: tpad_$mode_t;
    in x_scale: integer;
    in y_scale: integer;
    in hysteresis: integer;
    in origin: smd_$pos_t);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/tpad.ins.ftn'
```

```
integer*2 mode, x_scale, y_scale, hysteresis
integer*2 line, column, origin(2)
```

```
equivalence (line, origin(1)), (column, origin(2))
```

```
call tpad_$set_mode(mode, x_scale, y_scale, hysteresis, origin)
```

## DESCRIPTION

This call sets the mode, scale factors, and hysteresis factors of pointing devices. It can also change the origin for relative or absolute/relative mode. Use the output from `tpad_$inquire` as the input to this call to change one aspect of pointing device response without changing any other aspects.

A program can use this call with the touchpad, mouse, and bit pad pointing devices, but the mouse uses only the scale and hysteresis factors and ignores the other mode settings, since it is inherently a relative device.

*mode* The cursor mode to set. Specify one of the following predefined values:

**tpad\_\$absolute**

Absolute mode maps the pointing space to a part of the screen dictated by the scaling factor and the origin value. Thus, there is a one-to-one mapping between points in the pointing space and a subset of points on the screen.

**tpad\_\$relative**

In relative mode, the cursor responds only to device movement relative to the current position. The response of the cursor simulates response to mouse movement. This is the only meaningful mode to use with a mouse.

**tpad\_\$rel\_abs**

In relative/absolute mode, the cursor responds as in absolute mode on first contact to establish a new current position. After first contact, the cursor responds as in relative mode. Momentary lifting of the finger from a touchpad (for less than about half a second) does not affect the cursor position.

*x\_scale* The scale factor in the x dimension. Scale factors are not relevant to absolute mode.

*y\_scale* The scale factor in the y dimension. Scale factors are not relevant to absolute mode.

***hysteresis***

The hysteresis factor, in pixels.

The hysteresis factor prevents minor, unintentional movements from changing the cursor location. It effectively defines a box around the current location. The cursor does not move while the current location stays within the hysteresis box.

If the current location moves beyond the box, the distance the cursor moves is the difference between the new location and the past location less the hysteresis factor. The default hysteresis factor is 5.

*origin* is scale origin for x- and y-dimensions in **smd\_\$pos\_t** format.

**SEE ALSO**

**tpad\_\$inq\_dtype.**

---

**Contents**

intro .....	VEC-1
vec_\$add_constant .....	VEC-2
vec_\$add_constant_i .....	VEC-4
vec_\$add_vector .....	VEC-7
vec_\$add_vector_i .....	VEC-9
vec_\$copy .....	VEC-12
vec_\$copy_i .....	VEC-14
vec_\$dadd_constant .....	VEC-17
vec_\$dadd_constant_i .....	VEC-19
vec_\$dadd_vector .....	VEC-22
vec_\$dadd_vector_i .....	VEC-24
vec_\$dcopy .....	VEC-27
vec_\$dcopy_i .....	VEC-29
vec_\$ddot .....	VEC-32
vec_\$ddot_i .....	VEC-34
vec_\$dinit .....	VEC-37
vec_\$dmat_mult .....	VEC-39
vec_\$dmat_multn .....	VEC-41
vec_\$dmax .....	VEC-43
vec_\$dmax_i .....	VEC-46
vec_\$dmult_add .....	VEC-49
vec_\$dmult_add_i .....	VEC-51
vec_\$dmult_constant .....	VEC-54
vec_\$dmult_constant_i .....	VEC-56
vec_\$dot .....	VEC-59
vec_\$dot_i .....	VEC-61
vec_\$dp_sp .....	VEC-64
vec_\$dp_sp_i .....	VEC-66

---

---

vec_\$dpostmult .....	VEC-68
vec_\$dpostmultn .....	VEC-70
vec_\$dpremult .....	VEC-72
vec_\$dpremultn .....	VEC-74
vec_\$dsub .....	VEC-76
vec_\$dsub_i .....	VEC-78
vec_\$dsum .....	VEC-81
vec_\$dsum_i .....	VEC-83
vec_\$dswap .....	VEC-85
vec_\$dswap_i .....	VEC-87
vec_\$dzero .....	VEC-90
vec_\$dzero_i .....	VEC-92
vec_\$iadd_constant .....	VEC-94
vec_\$iadd_constant16 .....	VEC-96
vec_\$iadd_constant16_i .....	VEC-98
vec_\$iadd_constant_i .....	VEC-101
vec_\$iadd_vector .....	VEC-104
vec_\$iadd_vector16 .....	VEC-106
vec_\$iadd_vector16_i .....	VEC-108
vec_\$iadd_vector_i .....	VEC-111
vec_\$icopy .....	VEC-114
vec_\$icopy16 .....	VEC-116
vec_\$icopy16_i .....	VEC-118
vec_\$icopy_i .....	VEC-121
vec_\$idot .....	VEC-124
vec_\$idot16 .....	VEC-126
vec_\$idot16_i .....	VEC-128
vec_\$idot_i .....	VEC-131
vec_\$iinit .....	VEC-134
vec_\$iinit16 .....	VEC-136
vec_\$imat_mult .....	VEC-138
vec_\$imat_mult16 .....	VEC-140
vec_\$imat_multn .....	VEC-142
vec_\$imat_multn16 .....	VEC-144
vec_\$imax .....	VEC-146
vec_\$imax16 .....	VEC-149
vec_\$imax16_i .....	VEC-152
vec_\$imax_i .....	VEC-155
vec_\$imult_add .....	VEC-158
vec_\$imult_add16 .....	VEC-160
vec_\$imult_add16_i .....	VEC-162
vec_\$imult_add_i .....	VEC-165

---

---

vec_\$imult_constant .....	VEC-168
vec_\$imult_constant16 .....	VEC-170
vec_\$imult_constant16_i .....	VEC-172
vec_\$imult_constant_i .....	VEC-175
vec_\$init .....	VEC-178
vec_\$ipostmult .....	VEC-180
vec_\$ipostmult16 .....	VEC-182
vec_\$ipostmultn .....	VEC-184
vec_\$ipostmultn16 .....	VEC-186
vec_\$ipremult .....	VEC-188
vec_\$ipremult16 .....	VEC-190
vec_\$ipremultn .....	VEC-192
vec_\$ipremultn16 .....	VEC-194
vec_\$isub .....	VEC-196
vec_\$isub16 .....	VEC-198
vec_\$isub16_i .....	VEC-200
vec_\$isub_i .....	VEC-203
vec_\$isum .....	VEC-206
vec_\$isum16 .....	VEC-208
vec_\$isum16_i .....	VEC-210
vec_\$isum_i .....	VEC-212
vec_\$iswap .....	VEC-214
vec_\$iswap16 .....	VEC-216
vec_\$iswap16_i .....	VEC-218
vec_\$iswap_i .....	VEC-221
vec_\$izero .....	VEC-224
vec_\$izero16 .....	VEC-226
vec_\$izero16_i .....	VEC-228
vec_\$izero_i .....	VEC-230
vec_\$mat_mult .....	VEC-232
vec_\$mat_multn .....	VEC-234
vec_\$max .....	VEC-236
vec_\$max_i .....	VEC-239
vec_\$mult_add .....	VEC-242
vec_\$mult_add_i .....	VEC-244
vec_\$mult_constant .....	VEC-247
vec_\$mult_constant_i .....	VEC-249
vec_\$postmult .....	VEC-252
vec_\$postmultn .....	VEC-254
vec_\$premult .....	VEC-256
vec_\$premultn .....	VEC-258
vec_\$sp_dp .....	VEC-260

---

---

vec_\$sp_dp_i .....	VEC-262
vec_\$sub .....	VEC-264
vec_\$sub_i .....	VEC-266
vec_\$sum .....	VEC-269
vec_\$sum_i .....	VEC-271
vec_\$swap .....	VEC-273
vec_\$swap_i .....	VEC-275
vec_\$zero .....	VEC-278
vec_\$zero_i .....	VEC-280

**NAME**

intro – the Vector Library

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

**DESCRIPTION**

The Vector Library performs floating-point and integer vector matrix arithmetic.

Most of the vector calls have four versions: single-precision floating-point, double-precision floating-point, 16-bit integer, and 32-bit integer. The names of all single-precision vector calls begin with the simple prefix `vec_$`. Double-precision calls begin with the prefix `vec_$d`. The 16-bit integer calls begin with the prefix `vec_$i` and add a suffix of `16`. The 32-bit integer calls begin with the prefix `vec_$i`, but lack the `16` suffix. For example, `vec_$dot` and `vec_$ddot` are single- and double-precision versions of dot (scalar) product calls. `Vec_$idot` and `vec_$idot16` are the 32-bit and 16-bit integer versions, respectively.

Each variant of a routine takes similar arguments that differ from the arguments of the other variants only in the types of their operands. For the double-precision calls, all floating-point parameters are double-precision; for the single-precision calls, all floating-point parameters must be single precision; for the integer procedures and functions, the parameters and returned values are integers, etc.

Names that ultimately end in `_i` denote “incremental” calls, which step through vector arrays at user-specified increments. They are mainly useful for operations on vectors in matrixes that are not stored contiguously in memory.

**NOTES**

When using any of the vector calls, make sure that the indexes you pass are valid. In the interest of performance, these calls do not check index values for validity.

## NAME

`vec_$add_constant` – add a scalar to a single-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$add_constant(
    float *start_vec,
    long int &length,
    float &constant,
    float *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$add_constant(
    in start_vec: univ vec_$real_vector;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$real_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 256)

real *start\_vec*(*nvec*), *result\_vec*(*nvec*), *constant*  
integer\*4 *length*

call `vec_$add_constant(start_vec, length, constant, result_vec)`

## DESCRIPTION

`Vec_$add_constant` adds the scalar *constant* to the single-precision vector *start\_vec*, and supplies the result in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant + start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant + start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant + start_vec(i)
10 continue

```

*start\_vec*     The operand vector that *constant* will be added to.

*length*     The number of elements in *start\_vec* that *constant* will be added to.

*constant*     A scalar constant to be added to *start\_vec*.

*result\_vec*   The vector resulting from adding *constant* to *start\_vec*.

#### NOTES

When *vec\_\$add\_constant* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_constant\_i*, *vec\_\$dadd\_constant*, *vec\_\$iadd\_constant*, *vec\_\$iadd\_constant16*.

## NAME

`vec_$add_constant_i` – add a scalar to a vector in a single-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$add_constant_i(
    float *start_vec,
    long int &inc1,
    long int &length,
    float &constant,
    float *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$add_constant_i(
    in start_vec: univ vec_$real_vector;
    in inc1: integer32;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$real_vector;
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real start_vec(nvec), result_vec(nvec), constant
integer*4 length, inc1, inc2
```

```
call vec_$add_constant_i(start_vec, inc1, length,
& constant, result_vec, inc2)
```

## DESCRIPTION

`Vec_$add_constant_i` adds the scalar *constant* to elements of the single-precision array *start\_vec* selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$add_constant_i` to operate on individual vectors in a matrix. To add *constant* to the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the result of the operation in the

*N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant + start_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := constant + start_vec[j];
    k := k + inc2;
    j := j + incl;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
do 10 i = 1, length
    result_vec(k) = constant + start_vec(j)
    k = k + inc2
    j = j + incl
10 continue

```

*start\_vec*

The operand array whose elements will be summed with *constant*.

*incl*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with *constant*.

*length*

The number of elements in *start\_vec* that *constant* will be added to.

*constant*

The scalar value to be summed with elements of *start\_vec*.

*result\_vec*

The array resulting from summing *constant* with elements of *start\_vec*.

*inc2* Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums of *constant* with the elements of *start\_vec* selected by *incl*.

**NOTES**

In C and Pascal, *vec\_\$add\_constant\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$add\_constant*, *vec\_\$dadd\_constant\_i*, *vec\_\$iadd\_constant\_i*,  
*vec\_\$iadd\_constant16\_i*.

## NAME

`vec_$add_vector` – add two single-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$add_vector(
    float *start_vec,
    float *add_vec,
    long int &length,
    float *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$add_vector(
    in start_vec: univ vec_$real_vector;
    in add_vec: univ vec_$real_vector;
    in length: integer32;
    out result_vec: univ vec_$real_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real start_vec(nvec), add_vec(nvec), result_vec(nvec)
integer*4 length
```

call `vec_$add_vector(start_vec, add_vec, length, result_vec)`

## DESCRIPTION

`Vec_$add_vector` adds the single-precision vectors *start\_vec* and *add\_vec*, and supplies the vector sum in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] + add_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := start_vec[i] + add_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = start_vec(i) + add_vec(i)
10 continue

```

*start\_vec*

An addend vector.

*add\_vec*

An addend vector.

*length* Number of elements to sum. *Length* is usually just the order of the addends.

*result\_vec*

The vector that is the sum of *start\_vec* and *add\_vec*.

#### NOTES

When *vec\_\$add\_vector* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_vector\_i*, *vec\_\$dadd\_vector*, *vec\_\$iadd\_vector*, *vec\_\$iadd\_vector16*.

## NAME

vec\_\$add\_vector\_i – add vectors in two single-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$add_vector_i(
    float *start_vec,
    long int &inc1,
    float *add_vec,
    long int &inc2,
    long int &length,
    float *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$add_vector_i(
    in start_vec: univ vec_$real_vector;
    in inc1: integer32;
    in add_vec: univ vec_$real_vector;
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_$real_vector;
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real start_vec(nvec), add_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3
```

```
call vec_$add_vector_i(start_vec, inc1, add_vec, inc2,
    & length, result_vec, inc3)
```

## DESCRIPTION

Vec\_\$add\_vector\_i adds elements of the array *start\_vec*, selected by *inc1*, to elements of the array *add\_vec*, selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use *vec\_\$add\_vector\_i* to add individual vectors in two matrixes and place the sum in a

vector of another matrix. To add the  $M$ th vector in matrix  $X$  to the  $N$ th vector in matrix  $Y$ , choose  $inc1$  equal to the number of vectors in matrix  $X$  and  $inc2$  equal to the number of vectors in matrix  $Y$ . Then place the  $M$ th element of matrix  $X$  at the beginning of  $start\_vec$ , and place the  $N$ th element of matrix  $Y$  at the beginning of  $add\_vec$ . To place the result of the operation in the  $P$ th vector of a resultant matrix, choose  $inc3$  equal to the number of vectors in the resultant matrix, and place the  $P$ th element of the matrix array at the beginning of  $result\_vec$ .

In C, the resulting operation is

```

j = 0;
k = 0;
m = 0;
for (i = 0; i < length; ++i) {
    result_vec[j] = start_vec[k] + add_vec[m];
    j += inc3;
    k += inc1;
    m += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
m := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] + add_vec[m];
    j := j + inc3;
    k := k + inc1;
    m := m + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
m = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) + add_vec(m)
    j = j + inc3
    k = k + inc1
    m = m + inc2
10 continue

```

*start\_vec*

An array whose elements will be summed with elements of *add\_vec*.

*incl*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with elements of *add\_vec*.

*add\_vec*

An array whose elements will be summed with elements of *start\_vec*.

*inc2*

Increment for the index of *add\_vec* used to select the elements of *add\_vec* that will be summed with elements of *start\_vec*.

*length*

The number of elements in *start\_vec* that will be summed with elements of *add\_vec*.

*result\_vec*

An array to contain the *length* sums of elements of *start\_vec* and *add\_vec*.

*inc3*

Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums.

#### NOTES

In C and Pascal, *vec\_\$add\_vector\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

#### SEE ALSO

*vec\_\$add\_vector*, *vec\_\$dadd\_vector\_i*, *vec\_\$iadd\_vector16\_i*, *vec\_\$iadd\_vector\_i*.

## NAME

vec\_\$copy – copy a single-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$copy(
    float *start_vec,
    float *result_vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$copy(
    in start_vec: univ vec_$real_vector;
    out result_vec: univ vec_$real_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real start_vec(nvec), result_vec(nvec)
integer*4 length
```

```
call vec_$copy(start_vec, result_vec, length)
```

## DESCRIPTION

Vec\_\$copy copies *length* elements from *start\_vec* to *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    result_vec[i] := start_vec[i];
end
```

In FORTRAN, the resulting operation is

```
      do 10 i = 1, length
         result_vec(i) = start_vec(i)
10    continue
```

*start\_vec*

The vector that *result\_vec* will be copied from.

*result\_vec*

The vector that *start\_vec* will be copied to.

*length*

The number of elements to copy from *start\_vec* to *result\_vec*.

#### NOTES

When *vec\_\$copy* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$copy\_i*, *vec\_\$dcopy*, *vec\_\$icopy*, *vec\_\$icopy16*.

## NAME

`vec_$copy_i` – copy a vector from one single-precision matrix to another

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$copy_i(
    float *start_vec,
    long int &inc1,
    float *result_vec,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$copy_i(
    in start_vec: univ vec_$real_vector;
    in inc1: integer32;
    out result_vec: univ vec_$real_vector;
    in inc2: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *start\_vec*(*nvec*), *result\_vec*(*nvec*)  
integer\*4 *length*, *inc1*, *inc2*

call `vec_$copy_i`(*start\_vec*, *inc1*, *result\_vec*, *inc2*, *length*)

## DESCRIPTION

`Vec_$copy_i` copies *length* elements of the single-precision array *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$copy_i` to copy a vector from one matrix to another. To copy the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the copy into the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = start_vec[j];
    k += inc2;
    j += inc1;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
    begin
        result_vec[k] := start_vec[j];
        k := k + inc2;
        j := j + inc1;
    end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result_vec(j) = start_vec(k)
          j = j + inc2
          k = k + inc1
10    continue

```

*start\_vec*

The array whose elements will be copied to *result\_vec*.

*inc1*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be copied to *result\_vec*.

*result\_vec*

The array resulting from copying elements of *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

*inc2*

Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the copies of the elements of *start\_vec* selected by *inc1*.

*length*

The number of elements in *start\_vec* that will be copied to *result\_vec*.

VEC\_\$COPY\_I

VEC\_\$COPY\_I

**NOTES**

In C and Pascal, `vec_$copy_i` copies column vectors; whereas in FORTRAN, it copies row vectors.

**SEE ALSO**

`vec_$copy`, `vec_$dcopy_i`, `vec_$icopy16_i`, `vec_$icopy_i`.

## NAME

`vec_$(dadd)_constant` – add a scalar to a double-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(dadd)_constant(
    double *start_vec,
    long int &length,
    double &constant,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(dadd)_constant(
    in start_vec: univ vec_$(double)_vector;
    in length: integer32;
    in constant: double;
    out result_vec: univ vec_$(double)_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 256)
```

```
double precision start_vec(nvec)
double precision result_vec(nvec)
double precision constant
integer*4 length
```

```
call vec_$(dadd)_constant(start_vec, length, constant, result_vec)
```

## DESCRIPTION

`Vec_$(dadd)_constant` adds the scalar *constant* to the double-precision vector *start\_vec*, and supplies the result in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant + start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant + start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant + start_vec(i)
10 continue

```

*start\_vec*

The operand vector that *constant* will be added to.

*length* The number of elements in *start\_vec* that *constant* will be added to.

*constant*

A scalar constant to be added to *start\_vec*.

*result\_vec*

The vector resulting from adding *constant* to *start\_vec*.

#### NOTES

When *vec\_\$dadd\_constant* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_constant*, *vec\_\$add\_constant\_i*, *vec\_\$iadd\_constant*, *vec\_\$iadd\_constant16*.

## NAME

`vec_$dadd_constant_i` – add a scalar to a vector in a double-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dadd_constant_i(
    float *start_vec,
    long int &inc1,
    long int &length,
    float &constant,
    float *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dadd_constant_i(
    in start_vec: univ vec_$real_vector;
    in inc1: integer32;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$real_vector;
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real start_vec(nvec), result_vec(nvec), constant
integer*4 length, inc1, inc2
```

```
call vec_$dadd_constant_i(start_vec, inc1, length,
& constant, result_vec, inc2)
```

## DESCRIPTION

`Vec_$dadd_constant_i` adds the scalar *constant* to elements of the double-precision array *start\_vec* selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$dadd_constant_i` to operate on individual vectors in a matrix. To add *constant* to the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the result of the

operation in the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant + start_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
    begin
        result_vec[k] := constant + start_vec[j];
        k := k + inc2;
        j := j + incl;
    end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result_vec(k) = constant + start_vec(j)
          k = k + inc2
          j = j + incl
10  continue

```

*start\_vec*

The operand array whose elements will be summed with *constant*.

*incl*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with *constant*.

*length*

The number of elements in *start\_vec* that *constant* will be added to.

*constant*

The scalar value to be summed with elements of *start\_vec*.

*result\_vec*

The array resulting from summing *constant* with elements of *start\_vec*.

*inc2* Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums of *constant* with the elements of *start\_vec* selected by *incl*.

**NOTES**

In C and Pascal, *vec\_\$dadd\_constant\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$add\_constant*, *vec\_\$add\_constant\_i*, *vec\_\$iadd\_constant\_i*,  
*vec\_\$iadd\_constant16\_i*.

## NAME

`vec_$$dadd_vector` – add two double-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$dadd_vector(
    double *start_vec,
    double *add_vec,
    long int &length,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$dadd_vector(
    in start_vec: univ vec_$$double_vector;
    in add_vec: univ vec_$$double_vector;
    in length: integer32;
    out result_vec: univ vec_$$double_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double-precision *start\_vec*(*nvec*), *add\_vec*(*nvec*), *result\_vec*(*nvec*)  
integer\*4 *length*

call `vec_$$dadd_vector`(*start\_vec*, *add\_vec*, *length*, *result\_vec*)

## DESCRIPTION

`Vec_$$dadd_vector` adds the double-precision vectors *start\_vec* and *add\_vec* and supplies the vector sum in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] + add_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    result_vec[i] := start_vec[i] + add_vec[i];
end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i) + add_vec(i)
10 continue
```

*start\_vec*  
An addend vector.

*add\_vec*  
An addend vector.

*length* Number of elements to sum. *Length* is usually just the order of the addends.

*result\_vec*  
The vector that is the sum of *start\_vec* and *add\_vec*.

#### NOTES

When *vec\_\$dadd\_vector* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_vector*, *vec\_\$iadd\_vector*, *vec\_\$iadd\_vector16*.

## NAME

vec\_\$(dadd\_vector\_i – add vectors in two double-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$(dadd_vector_i(
    double *start_vec,
    long int &inc1,
    double *add_vec,
    long int &inc2,
    long int &length,
    double *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$(dadd_vector_i(
    in start_vec: univ vec_$(double_vector);
    in inc1: integer32;
    in add_vec: univ vec_$(double_vector);
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_$(double_vector);
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

double precision start_vec(nvec), add_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3

call vec_$(dadd_vector_i(start_vec, inc1, add_vec, inc2,
& length, result_vec, inc3)
```

## DESCRIPTION

Vec\_\$(dadd\_vector\_i adds elements of the array *start\_vec*, selected by *inc1*, to elements of the array *add\_vec*, selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use vec\_\$(dadd\_vector\_i to add individual vectors in two matrixes and place the sum in a

vector of another matrix. To add the  $M$ th vector in matrix  $X$  to the  $N$ th vector in matrix  $Y$ , choose  $inc1$  equal to the number of vectors in matrix  $X$  and  $inc2$  equal to the number of vectors in matrix  $Y$ . Then place the  $M$ th element of matrix  $X$  at the beginning of  $start\_vec$ , and place the  $N$ th element of matrix  $Y$  at the beginning of  $add\_vec$ . To place the result of the operation in the  $P$ th vector of a resultant matrix, choose  $inc3$  equal to the number of vectors in the resultant matrix, and place the  $P$ th element of the matrix array at the beginning of  $result\_vec$ .

In C, the resulting operation is

```

j = 0;
k = 0;
m = 0;
for (i = 0; i < length; ++i) {
    result_vec[j] = start_vec[k] + add_vec[m];
    j += inc3;
    k += inc1;
    m += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
m := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] + add_vec[m];
    j := j + inc3;
    k := k + inc1;
    m := m + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
m = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) + add_vec(m)
    j = j + inc3
    k = k + inc1
    m = m + inc2
10 continue

```

*start\_vec*

An array whose elements will be summed with elements of *add\_vec*.

*inc1*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with elements of *add\_vec*.

*add\_vec*

An array whose elements will be summed with elements of *start\_vec*.

*inc2*

Increment for the index of *add\_vec* used to select the elements of *add\_vec* that will be summed with elements of *start\_vec*.

*length*

The number of elements in *start\_vec* that will be summed with elements of *add\_vec*.

*result\_vec*

An array to contain the *length* sums of elements of *start\_vec* and *add\_vec*.

*inc3*

Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums.

## NOTES

In C and Pascal, *vec\_\$dadd\_vector\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

## SEE ALSO

*vec\_\$add\_vector\_i*, *vec\_\$iadd\_vector16\_i*, *vec\_\$iadd\_vector\_i*.

## NAME

`vec_$dcopy` – copy a double-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$dcopy(
    double *start_vec,
    double *result_vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$dcopy(
    in start_vec: univ vec_$double_vector;
    out result_vec: univ vec_$double_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

double precision start_vec(nvec), result_vec(nvec)
integer*4 length

call vec_$dcopy(start_vec, result_vec, length)
```

## DESCRIPTION

`Vec_$dcopy` copies *length* elements from *start\_vec* to *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    begin
        result_vec[i] := start_vec[i];
    end
```

In FORTRAN, the resulting operation is

```
      do 10 i = 1, length
         result_vec(i) = start_vec(i)
10    continue
```

*start\_vec*

The vector that *result\_vec* will be copied from.

*result\_vec*

The vector that *start\_vec* will be copied to.

*length*

The number of elements to copy from *start\_vec* to *result\_vec*.

#### NOTES

When *vec\_\$dcopy* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$copy*, *vec\_\$dcopy\_i*, *vec\_\$icopy*, *vec\_\$icopy16*.

## NAME

`vec_$dcopy_i` – copy a vector from one double-precision matrix to another

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dcopy_i(
    double *start_vec,
    long int &inc1,
    double *result_vec,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dcopy_i(
    in start_vec: univ vec_$double_vector;
    in inc1: integer32;
    out result_vec: univ vec_$double_vector;
    in inc2: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *start\_vec*(*nvec*), *result\_vec*(*nvec*)  
integer\*4 *length*, *inc1*, *inc2*

call `vec_$dcopy_i`(*start\_vec*, *inc1*, *result\_vec*, *inc2*, *length*)

## DESCRIPTION

`Vec_$dcopy_i` copies *length* elements of the double-precision array *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$dcopy_i` to copy a vector from one matrix to another. To copy the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the copy into the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = start_vec[j];
    k += inc2;
    j += inc1;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
    begin
        result_vec[k] := start_vec[j];
        k := k + inc2;
        j := j + inc1;
    end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result_vec(j) = start_vec(k)
          j = j + inc2
          k = k + inc1
10    continue

```

*start\_vec*

The array whose elements will be copied to *result\_vec*.

*inc1*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be copied to *result\_vec*.

*result\_vec*

The array resulting from copying elements of *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

*inc2*

Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the copies of the elements of *start\_vec* selected by *inc1*.

*length*

The number of elements in *start\_vec* that will be copied to *result\_vec*.

VEC\_\$DCOPY\_I

VEC\_\$DCOPY\_I

**NOTES**

In C and Pascal, `vec_$dcopy_i` copies column vectors; whereas in FORTRAN, it copies row vectors.

**SEE ALSO**

`vec_$copy_i`, `vec_$dcopy`, `vec_$icopy16_i`, `vec_$icopy_i`.

## NAME

`vec_$$ddot` – return the dot product of two double-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
double vec_$$ddot(
    double *vector1,
    double *vector2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$$ddot(
    in vector1: univ vec_$$double_vector;
    in vector2: univ vec_$$double_vector;
    in length: integer32): double;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double-precision *vector1*(*nvec*), *vector2*(*nvec*), *result*  
integer\*4 *length*

*result* = `vec_$$ddot`(*vector1*, *vector2*, *length*)

## DESCRIPTION

`Vec_$$ddot` returns the dot (scalar) product of two single-precision vectors, *vector1* and *vector1*.

In C, the resulting operation is

```
return_value = 0.0;
for (i = 0; i < length; ++i)
    return_value += vector1[i] * vector2[i];
```

In Pascal, the resulting operation is

```
return_value := 0.0;
for i := 1 to length do
begin
  return_value := return_value
                + vector1[i] * vector2[i];
end
```

In FORTRAN, the resulting operation is

```
vec_$ddot = 0.0
do 10 i = 1,length
  vec_$ddot = vec_$ddot + vector1(i) * vector2(i)
10 continue
vector1 A vector.
vector2 Another vector.
length The number of elements to use in calculating the dot (scalar) product.
```

#### NOTES

When `vec_$ddot` is used on matrixes in C or Pascal, `vector1` and `vector2` are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

`vec_$ddot_i`, `vec_$dot`, `vec_$idot`, `vec_$idot16`.

**NAME**

**vec\_\$\$ddot\_i** – return the dot product of two vectors in double-precision matrixes

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
double vec_$$ddot_i(
    double *vector1,
    long int &incl1,
    double *vector2,
    long int &inc2,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$$ddot_i(
    in vector1: univ vec_$$double_vector;
    in incl1: integer32;
    in vector2: univ vec_$$double_vector;
    in inc2: integer32;
    in length: integer32): double;
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
double precision vector1(nvec), vector2(nvec), result
integer*4 length, incl1, inc2
```

```
result = vec_$$ddot_i(vector1, incl1, vector2, inc2, length)
```

**DESCRIPTION**

**Vec\_\$\$ddot\_i** returns the dot (scalar) product of two vectors from the double-precision arrays *vector1* and *vector2*.

In C, the resulting operation is

```

j = 0;
k = 0;
return_value = 0.0;
for (i = 0; i < length; ++i) {
    return_value += vector1[j] * vector2[k];
    j += inc1;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
return_value := 0.0;
for i := 1 to length do
    begin
        return_value := return_value
            + vector1[j] * vector2[k];
        j := j + inc1;
        k := k + inc2;
    end;

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      vec_$ddot_i = 0.0
      do 10 i = 1,length
          vec_$ddot_i = vec_$ddot_i
&              + vector1(j) * vector2(k)
              j = j + inc1
              k = k + inc2
10  continue

```

*vector1* An array containing one of the vectors to use in calculating the dot product.

*inc1* An increment for *vector1* that chooses which elements will be used to calculate the product.

*vector2* An array containing the other vector to use in calculating the dot product.

*inc2* An increment for *vector2* that chooses which elements will be used to calculate the product.

*length* The number of elements from *vector1* or *vector2* to use in calculating the dot product.

**NOTES**

When `vec_$$dot_i` is used on matrixes in C or Pascal, *vector1* and *vector2* are column vectors; whereas in FORTRAN, they are row vectors.

**SEE ALSO**

`vec_$$dot`, `vec_$$dot_i`, `vec_$$dot16_i`, `vec_$$dot_i`.

## NAME

`vec_$dinit` – initialize a double-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dinit(
    double *vector,
    long int &length,
    double &constant)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dinit(
    var vector: univ vec_$double_vector;
    in length: integer32;
    in constant: double);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *vector*(*nvec*), *constant*  
integer\*4 *length*

call `vec_$dinit`(*vector*, *length*, *constant*)

## DESCRIPTION

`Vec_$dinit` sets *length* elements of *vector* to the double-precision value *constant*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = constant;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    begin
        vector[i] := constant;
    end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    vector(i) = constant
10 continue
```

*vector* The vector to initialize.

*length* The number of elements in *vector* to initialize.

*constant*

The value that the elements of *vector* should be set to.

#### NOTES

In C and Pascal, `vec_$dinit` initializes a row vector; whereas in FORTRAN, it initializes a column vector.

#### SEE ALSO

`vec_$dinit_i`, `vec_$iinit`, `vec_$iinit16`, `vec_$init`.

## NAME

`vec_$dmat_mult` – multiply two 4x4 double-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dmat_mult(
    double *matrix1,
    double *matrix2,
    double *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dmat_mult(
    in matrix1: univ vec_$double_matrix;
    in matrix2: univ vec_$double_matrix;
    out out_matrix: univ vec_$double_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
double precision matrix1(4, 4), matrix2(4, 4), out_matrix(4, 4)
```

```
call vec_$dmat_mult(matrix1, matrix2, out_matrix)
```

## DESCRIPTION

`Vec_$dmat_mult` multiplies two 4x4 matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, `vec_$dmat_mult` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```
for (i = 0; i < 4; ++i)
    for (j = 0; j < 4; ++j) {
        out_mat[j,i] = 0.0;
        for (k = 0; k < 4; ++k)
            out_mat[j][i] += matrix1[k][i]
                * matrix2[j][k];
    }
```

In Pascal, `vec_$dmat_mult` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to 4 do
  for j := 1 to 4 do
    begin
      out_mat[j,i] := 0.0;
      for k := 1 to 4 do
        out_mat[j,i] := out_mat[j,i]
          + matrix1[k,i]
          * matrix2[j,k];
      end;
    end;
  end;
end;

```

In FORTRAN, `vec_$dmat_mult` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, 4
  do 10 j = 1, 4
    out_mat(i,j) = 0.0
    do 10 k = 1, 4
      out_mat(i,j) = out_mat(i,j)
        &               + matrix1(i,k)
        &               * matrix2(k,j)
    10 continue

```

*matrix1* A 4x4 matrix.

*matrix2* Another 4x4 matrix.

*out\_matrix*

The product of *matrix1* and *matrix2*.

#### NOTES

`Vec_$dmat_multn` performs the same operations for variably dimensioned matrices.

#### SEE ALSO

`vec_$imat_mult`, `vec_$imat_mult16`, `vec_$mat_mult`.

## NAME

vec\_\$dmat\_multn – multiply two double-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dmat_multn(
    double *matrix1,
    double *matrix2,
    long int &m,
    long int &n,
    long int &s,
    double *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dmat_multn(
    in matrix1: univ vec_$double_matrix;
    in matrix2: univ vec_$double_matrix;
    in m: integer32;
    in n: integer32;
    in s: integer32;
    out out_matrix: univ vec_$double_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n, s
parameter (m = 3, n = 4, s = 5)

double precision matrix1(m, n), matrix2(n, s), out_matrix(m, s)

call vec_$dmat_multn(matrix1, matrix2, m, n, s, out_matrix)
```

## DESCRIPTION

Vec\_\$dmat\_multn multiplies two variably dimensioned matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, vec\_\$dmat\_multn calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for (i = 0; i < m; ++i)
  for (j = 0; j < s; ++j) {
    out_matrix[j][i] = 0.0;
    for (k = 0; k < n; ++k)
      out_matrix[j][i] += matrix1[k][i]
                        * matrix2[j][k];
  }

```

In Pascal, `vec_$dmat_multn` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to m do
  for j := 1 to s do
    begin
      out_matrix[j,i] = 0.0;
      for k := 1 to n do
        out_matrix[j,i] := out_matrix[j,i]
                          + matrix1[k,i]
                          * matrix2[j,k];
      end;
    end;

```

In FORTRAN, `vec_$dmat_multn` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, m
  do 10 j = 1, s
    out_matrix(i,j) = 0.0
    do 10 k = 1, n
      out_matrix(i,j) = out_matrix(i,j)
                      + matrix1(i,k)
                      * matrix2(k,j)
    &
    &
  10 continue

```

*matrix1* A matrix to be multiplied.

*matrix2* Another matrix to be multiplied.

*m, n, s* The various matrix dimensions.

*out\_matrix*

The product of *matrix1* and *matrix2*.

#### SEE ALSO

`vec_$imat_multn`, `vec_$imat_multn16`, `vec_$mat_multn`.

## NAME

`vec_$(d)max` – find the maximum absolute value in a double-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(d)max(
    double *vector,
    long int &length,
    double *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(d)max(
    in vector: univ vec_$(double)_vector;
    in length: integer32;
    out result: double;
    out location: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *vector*(*nvec*), *result*  
integer\*4 *length*, *location*

call `vec_$(d)max`(*vector*, *length*, *result*, *location*)

## DESCRIPTION

`Vec_$(d)max` searches through *length* elements of *vector* and supplies the value and location of the element with the greatest absolute value.

In C, the resulting operation is

```
result = fabs(vector[0]);
location = 1;
for (i = 1; i < length; ++i)
    if (fabs(vector[i]) > result) {
        location = i + 1;
        result = fabs(vector[i]);
    }
```

In Pascal, the resulting operation is

```
result := abs(vector[1]);
location = 1;
for 10 i := 2 to length do
    if (abs(vector[i]) > result) then
        begin
            location := i;
            result := abs(vector[i]);
        end
```

In FORTRAN, the resulting operation is

```
result = dabs(vector(1))
location = 1
do 10 i = 2, length
    if (dabs(vector(i)) .gt. result) then
        location = i
        result = dabs(vector(i))
    endif
10 continue
```

*vector* The vector to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, *vec\_\$dmax* searches a row vector; whereas in FORTRAN, it searches a column vector.

VEC\_\$DMAX

VEC\_\$DMAX

**SEE ALSO**

vec\_\$dmax\_i, vec\_\$imax, vec\_\$imax16, vec\_\$max.

## NAME

`vec_$dmax_i` – find the maximum absolute value in a vector from a double-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dmax_i(
    double *vector,
    long int &inc,
    long int &length,
    double *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dmax_i
  in vector: univ vec_$double_vector;
  in inc: integer32;
  in length: integer32;
  out result: double;
  out location: integer32;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *vector*(*nvec*), *result*  
integer\*4 *length*, *inc*, *location*

call `vec_$dmax_i`(*vector*, *inc*, *length*, *result*, *location*)

## DESCRIPTION

`Vec_$dmax_i` searches through the *length* elements of *vector* selected by *inc*, and supplies the value and location of the element with the greatest absolute value.

Through appropriate choice of *inc*, a program can use `vec_$dmax_i` to search a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

result = fabs(vector[0]);
location = 1;
j = inc;
for (i = 1; i < length, ++i) {
    if (fabs(vector[j]) > result) {
        location = i + 1;
        result = fabs(vector[j]);
    }
    j += inc;
}

```

In Pascal, the resulting operation is

```

result := abs(vector[1]);
location := 1;
j := 1 + inc;
for 10 i := 2 to length do
    begin
        if (abs(vector[j]) > result) then
            begin
                location := i;
                result := abs(vector[j]);
            end
        j := j + inc;
    end

```

In FORTRAN, the resulting operation is

```

result = dabs(vector(1))
location = 1
j = 1 + inc
do 10 i = 2, length
    if (dabs(vector(j)) .gt. result) then
        location = i
        result = dabs(vector(j))
    endif
    j = j + inc
10 continue

```

*vector* The array to search.

*inc* An increment for the index of *vector* that selects the elements to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, `vec_$dmax_i` searches a column vector; whereas in FORTRAN, it searches a row vector.

#### SEE ALSO

`vec_$dmax`, `vec_$imax16_i`, `vec_$imax_i`, `vec_$max_i`.

## NAME

`vec_$(mult)_add` – scale and add one double-precision vector to another

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(mult)_add(
    double *add_vec,
    double *mult_vec,
    long int &length,
    double &constant,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(mult)_add(
    in add_vec: univ vec_$(double)_vector;
    in mult_vec: univ vec_$(double)_vector;
    in length: integer32;
    in constant: double;
    out result_vec: univ vec_$(double)_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
double precision add_vec(nvec), mult_vec(nvec), result_vec(nvec), constant
integer*4 length
```

```
call vec_$(mult)_add(add_vec, mult_vec, length,
& constant, result_vec)
```

## DESCRIPTION

`vec_$(mult)_add` multiplies the vector `mult_vec` by the scalar `constant`, adds the product to the vector `add_vec`, and supplies the resulting vector in `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = add_vec[i]
        + constant * mult_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
    result_vec[i] := add_vec[i]
                    + constant * mult_vec[i];

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
    result_vec(i) = add_vec(i)
&                + constant * mult_vec(i)
10 continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*length*

The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

When *vec\_\$dmult\_add* is used to operate on matrixes in C and Pascal, *add\_vec*, *mult\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_add\_i*, *vec\_\$simult\_add*, *vec\_\$simult\_add16*, *vec\_\$mult\_add*.

## NAME

vec\_\$dmult\_add\_i – scale and add double-precision vectors in matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dmult_add_i(
    double *add_vec,
    long int &inc1,
    double *mult_vec,
    long int &inc2,
    long int &length,
    double &constant,
    double *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dmult_add_i(
    in add_vec: univ vec_$double_vector;
    in inc1: integer32;
    in mult_vec: univ vec_$double_vector;
    in inc2: integer32;
    in length: integer32;
    in constant: double;
    out result_vec: univ vec_$double_vector;
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

double precision add\_vec(nvec), mult\_vec(nvec), result\_vec(nvec), constant  
integer\*4 length, inc1, inc2, inc3

call vec\_\$dmult\_add\_i(add\_vec, inc1, mult\_vec, inc2, length,  
& constant, result\_vec, inc3)

## DESCRIPTION

Vec\_\$dmult\_add\_i multiplies *length* elements of *mult\_vec* selected by *inc2* by the scalar *constant*, adds the resulting products to elements of *add\_vec* selected by *inc1*, and supplies the results in elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *incl*, *inc2*, and *inc3*, a program can use `vec_$dmult_add_i` to scale and sum individual vectors in two matrixes and place the result in a vector of another matrix. To scale and add the *N*th vector in matrix *Y* to the *M*th vector in matrix *X*, choose *inc2* equal to the number of vectors in matrix *Y* and *incl* equal to the number of vectors in matrix *X*. Then place the *M*th element of matrix *X* at the beginning of *add\_vec*, and place the *N*th element of matrix *Y* at the beginning of *mult\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc3* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length; ++i) {
    result_vec[l] = add_vec[j]
                  + constant * mult_vec[k];
    j += incl;
    k += inc2;
    l += inc3;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[l] := add_vec[j]
                   + constant * mult_vec[k];
    j := j + incl;
    k := k + inc2;
    l := l + inc3;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      l = 1
      do 10 i = 1, length
         result(l) = add_vec(j)
      &          + constant * mult_vec(k)
         j = j + inc1
         k = k + inc2
         l = l + inc3
      10  continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*inc1*

An increment for the index of *add\_vec* that selects the elements to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*inc2*

An increment for the index of *mult\_vec* that selects the elements to multiply by *constant* and add to *add\_vec*.

*length*

The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

*inc3*

An increment for the index of *result\_vec* that selects the elements to receive the results of multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

In C and Pascal, *vec\_\$dmult\_add\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

#### SEE ALSO

*vec\_\$dmult\_add*, *vec\_\$simult\_add16\_i*, *vec\_\$simult\_add\_i*, *vec\_\$mult\_add\_i*.

## NAME

`vec_$(dmult)_constant` – multiply a double-precision vector by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$(dmult)_constant(
    double *mult_vec,
    long int &length,
    double &constant,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$(dmult)_constant(
    in mult_vec: univ vec_$(double)_vector;
    in length: integer32;
    in constant: double;
    out result_vec: univ vec_$(double)_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

double precision mult_vec(nvec), result_vec(nvec), constant
integer*4 length

call vec_$(dmult)_constant(mult_vec, length, constant, result_vec)
```

## DESCRIPTION

`Vec_$(dmult)_constant` multiplies the double-precision array `mult_vec` by the scalar value `constant` and supplies the result in the array `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant * start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant * start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant * mult_vec(i)
10 continue

```

*mult\_vec*

The vector to multiply by *constant*.

*length*

The number of elements to multiply.

*constant*

The scalar constant to multiply *mult\_vec* by.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant*.

#### NOTES

When *vec\_\$dmult\_constant* is used to operate on matrixes in C and Pascal, *mult\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_constant\_i*,  
*vec\_\$mult\_constant*.

*vec\_\$imult\_constant*,

*vec\_\$imult\_constant16*,

## NAME

`vec_$(dmult_constant_i` – multiply a vector in a double-precision matrix by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$(dmult_constant_i(
    double *mult_vec,
    long int &inc1,
    long int &length,
    double &constant,
    double *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$(dmult_constant_i(
    in mult_vec: univ vec_$(double_vector);
    in inc1: integer32;
    in length: integer32;
    in constant: double;
    out result_vec: univ vec_$(double_vector);
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

double precision mult_vec(nvec), result_vec(nvec), constant
integer*4 length, inc1, inc2

call vec_$(dmult_constant_i(mult_vec, inc1, length, constant,
& result_vec, inc2)
```

## DESCRIPTION

`Vec_$(dmult_constant_i` multiplies *length* elements of the double-precision array *mult\_vec* selected by *inc1* by the scalar value *constant*, and supplies the result in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$(dmult_constant_i` to operate on a vector in a matrix. To multiply *M*th vector in a matrix by *constant*, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *mult\_vec*. To place the result of the operation in the

*N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant * mult_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
    begin
        result_vec[k] := constant * mult_vec[j];
        k := k + inc2;
        j := j + incl;
    end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result(j) = constant * mult_vec(k)
          j = j + inc2
          k = k + incl
10    continue

```

*mult\_vec*

The vector to multiply by *constant*.

*incl*

An increment for the index of the array *mult\_vec* that selects elements to multiply by *constant*.

*length*

The number of products to calculate.

*constant*

The scalar constant to multiply elements of *mult\_vec* by.

*result\_vec*

An array whose elements receive the product of *constant* and *mult\_vec*.

VEC\_\$DMULT\_CONSTANT\_I

VEC\_\$DMULT\_CONSTANT\_I

*inc2* An increment for the index of the array *result\_vec* that selects elements to receive the product of *constant* and *mult\_vec*.

**NOTES**

In C and Pascal, *vec\_\$dmult\_constant\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$dmult\_constant*, *vec\_\$imult\_constant16\_i*, *vec\_\$imult\_constant\_i*,  
*vec\_\$mult\_constant\_i*.

## NAME

`vec_$dot` – return the dot product of two single-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
float vec_$dot(
    float *vector1,
    float *vector2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$dot(
    in vector1: univ vec_$real_vector;
    in vector2: univ vec_$real_vector;
    in length: integer32): real;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real vector1(nvec), vector2(nvec), result
integer*4 length
```

*result* = `vec_$dot(vector1, vector2, length)`

## DESCRIPTION

`Vec_$dot` returns the dot (scalar) product of two single-precision vectors, *vector1* and *vector2*.

In C, the resulting operation is

```
return_value = 0.0;
for (i = 0; i < length; ++i)
    return_value += vector1[i] * vector2[i];
```

In Pascal, the resulting operation is

```
return_value := 0.0;
for i := 1 to length do
  begin
    return_value := return_value
                  + vector1[i] * vector2[i];
  end
```

In FORTRAN, the resulting operation is

```
vec_$dot = 0.0
do 10 i = 1, length
  vec_$dot = vec_$dot + vector1(i) * vector2(i)
10 continue
```

*vector1* A vector.

*vector2* Another vector.

*length* The number of elements to use in calculating the dot (scalar) product.

#### NOTES

When *vec\_\$dot* is used on matrixes in C or Pascal, *vector1* and *vector2* are row vectors; whereas in FORTRAN they are column vectors.

#### SEE ALSO

*vec\_\$ddot*, *vec\_\$dot\_i*, *vec\_\$idot*, *vec\_\$idot16*.

## NAME

`vec_$dot_i` – return the dot product of two vectors in single-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
float vec_$dot_i(
    float *vector1,
    long int &inc1,
    float *vector2,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$dot_i(
    in vector1: univ vec_$real_vector;
    in inc1: integer32;
    in vector2: univ vec_$real_vector;
    in inc2: integer32;
    in length: integer32): real;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *vector1*(*nvec*), *vector2*(*nvec*), *result*  
integer\*4 *length*, *inc1*, *inc2*

*result* = `vec_$dot_i`(*vector1*, *inc1*, *vector2*, *inc2*, *length*)

## DESCRIPTION

`Vec_$dot_i` returns the dot (scalar) product of two vectors from the single-precision arrays *vector1* and *vector2*.

In C, the resulting operation is

```

j = 0;
k = 0;
return_value = 0.0;
for (i = 0; i < length; ++i) {
    return_value += vector1[j] * vector2[k];
    j += inc1;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
return_value := 0.0;
for i := 1 to length do
begin
    return_value := return_value
                    + vector1[j] * vector2[k];
    j := j + inc1;
    k := k + inc2;
end;

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      vec_$dot_i = 0.0
      do 10 i = 1,length
          vec_$dot_i = vec_$dot_i
&                + vector1(j) * vector2(k)
                j = j + inc1
                k = k + inc2
10  continue

```

*vector1* An array containing one of the vectors to use in calculating the dot product.

*inc1* An increment for *vector1* that chooses which elements will be used to calculate the product.

*vector2* An array containing the other vector to use in calculating the dot product.

*inc2* An increment for *vector2* that chooses which elements will be used to calculate the product.

*length* The number of elements from *vector1* or *vector2* to use in calculating the dot product.

**NOTES**

When `vec_dot_i` is used on matrixes in C or Pascal, *vector1* and *vector2* are column vectors; whereas in FORTRAN, they are row vectors.

**SEE ALSO**

`vec_ddot_i`, `vec_dot`, `vec_sidot16_i`, `vec_sidot_i`.

## NAME

`vec_$dp_sp` – copy a double-precision vector to a single-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dp_sp(
    double *dp_vec,
    float *sp_vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dp_sp(
    in dp_vec: univ vec_$double_vector;
    in sp_vec: univ vec_$real_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real sp_vec(nvec)
double precision dp_vec(nvec)
integer*4 length
```

call `vec_$dp_sp(dp_vec, sp_vec, length)`

## DESCRIPTION

`Vec_$dp_sp` copies *length* elements from the double-precision vector *dp\_vec* to the single-precision vector *sp\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    sp_vec[i] = (float)dp_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
  begin
    sp_vec[i] := dp_vec[i];
  end
```

In FORTRAN, the resulting operation is

```
do 10 i=1, length
  sp_vec(i) = sngl(dp_vec(i))
10 continue
```

*dp\_vec* The double-precision vector to copy from.  
*sp\_vec* The single-precision vector to copy to.  
*length* The number of elements to copy.

#### NOTES

In C and Pascal, `vec_$dp_sp` copies a row vector; whereas in FORTRAN, it copies a column vector.

#### SEE ALSO

`vec_$dp_sp_i`, `vec_$sp_dp`.

**NAME**

`vec_$dp_sp_i` – copy a vector from a double-precision matrix into a single-precision matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dp_sp_i(
    double *dp_vec,
    long int &inc1,
    float *sp_vec,
    long int &inc2,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dp_sp_i(
    in dp_vec: univ vec_$double_vector;
    in inc1: integer32;
    in sp_vec: univ vec_$real_vector;
    in inc2: integer32;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real sp_vec(nvec)
double precision dp_vec(nvec)
integer*4 length, inc1, inc2
```

call `vec_$dp_sp_i(dp_vec, inc1, sp_vec, inc2, length)`

**DESCRIPTION**

`Vec_$dp_sp_i` copies elements from a double-precision array *dp\_vec* selected by *inc1* to elements of a single-precision array *sp\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$dp_sp_i` to copy a vector from one matrix to another. To copy the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *dp\_vec*. To place the copy into the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *sp\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    sp_vec[i] = (float)dp_vec[i];
    j += incl;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    sp_vec[i] := dp_vec[i];
    j := j + incl;
    k := k + inc2;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          sp_vec(k) = sngl(dp_vec(j))
          j = j + incl
          k = k + inc2
10    continue

```

*dp\_vec* The double-precision array to copy from.

*incl* An increment for the index of *dp\_vec* that selects the elements to copy from.

*sp\_vec* The single-precision array to copy to.

*inc2* An increment for the index of *sp\_vec* that selects the elements to copy to.

*length* The number of elements to copy from *dp\_vec* to *sp\_vec*.

#### NOTES

In C and Pascal, *vec\_\$dp\_sp\_i* copies a column vector; whereas in FORTRAN, it copies a row vector.

#### SEE ALSO

*vec\_\$dp\_sp*, *vec\_\$sp\_dp\_i*.

## NAME

`vec_$$dpostmult` – multiply a double-precision vector by a 4x4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$dpostmult(
    double *matrix,
    double *start_vec,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$dpostmult(
    in matrix: univ vec_$$double_matrix;
    in start_vec: univ vec_$$double_vector;
    out result_vec: univ vec_$$double_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
double precision matrix(4, 4), start_vec(4), result_vec(4)
```

```
call vec_$$dpostmult(matrix, start_vec, result_vec)
```

## DESCRIPTION

`Vec_$$dpostmult` multiplies the 4-element vector `start_vec` by the 4x4 matrix `matrix`.

In C, `vec_$$dpostmult` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```
for (j = 0; j < 4; ++j) {
    result_vec[j] = 0.0;
    for (i = 0; i < 4; ++i)
        result_vec[j] += start_vec[i]
            * matrix[i][j];
}
```

In Pascal, `vec_$$dpostmult` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```

for j := 1 to 4 do
  begin
    result_vec[j] := 0.0;
    for i := 1 to 4 do
      result_vec[j] := result_vec[j]
        + start_vec[i]
        * matrix[i,j];
    end
  end
end

```

In FORTRAN, `vec_$dpostmult` applies *matrix* as a left transform to a column vector *start\_vec*, and the resulting operation is

```

do 10 j = 1, 4
  result_vec(j) = 0.0
  do 10 i = 1, 4
    result_vec(j) = result_vec(j)
      & + start_vec(i)
      & * matrix(j,i)
  10 continue

```

*matrix* The matrix to multiply by *start\_vec*.

*start\_vec* The vector to multiply by *matrix*.

*result\_vec* The product of *start\_vec* and *matrix*.

## NOTES

`Vec_$dpremult` transforms double-precision vectors from the other side.

## SEE ALSO

`vec_$ipostmult`, `vec_$ipostmult16`, `vec_$postmult`.

**NAME**

`vec_$_dpostmultn` – multiply a double-precision vector by a matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$_dpostmultn(
    double *matrix,
    double *start_vec,
    long int &m,
    long int &n,
    double *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$_dpostmultn(
    in matrix: univ vec_$_double_matrix;
    in start_vec: univ vec_$_double_vector;
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$_double_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
double precision matrix(m, n), start_vec(n), result_vec(m)
```

```
call vec_$_dpostmultn(matrix, start_vec, m, n, result_vec)
```

**DESCRIPTION**

`Vec_$_dpostmultn` multiplies the  $n$ -element vector `start_vec` by the variably dimensioned matrix `matrix`, and supplies the resulting  $m$ -element vector in `result_vec`.

In C, `vec_$_dpostmultn` applies the  $n \times m$  matrix `matrix` as a right transform to the  $m$ -element row vector `start_vec`, and supplies the transformed  $n$ -element result in `result_vec`:

```

for (i = 0; i < m; ++i) {
    result_vec[i] = 0.0;
    for (j = 0; j < n; ++j)
        result_vec[i] += start_vec[j]
                        * matrix[j][i];
}

```

In Pascal, `vec_$$dpostmultn` applies the  $n \times m$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to m do
begin
    result_vec[i] := 0.0;
    for j := 1 to n do
        result_vec[i] := result_vec[i]
                        + start_vec[j]
                        * matrix[j,i];
    end;
end;

```

In FORTRAN, `vec_$$dpostmultn` applies the  $m \times n$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, m
    result_vec(i) = 0.0
    do 10 j = 1, n
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(i,j)
    &
    &
10 continue

```

*matrix* A matrix to multiply *start\_vec* by.

*start\_vec* An  $n$ -element vector to multiply by *matrix*.

$m$  The number of elements in *start\_vec*.

$n$  The number of elements in *result\_vec*.

*result\_vec* An  $m$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$$dpremultn` transforms double-precision vectors from the other side.

#### SEE ALSO

`vec_$$ipostmultn`, `vec_$$ipostmultn16`, `vec_$$postmultn`.

## NAME

`vec_$$dpremult` – multiply a double-precision vector by a 4x4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$dpremult(
    double *start_vec,
    double *matrix,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$dpremult(
    in start_vec: univ vec_$$double_vector;
    in matrix: univ vec_$$double_matrix;
    out result_vec: univ vec_$$double_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
double precision start_vec(4), matrix(4,4), result_vec(4)
```

```
call vec_$$dpremult(start_vec, matrix, result_vec)
```

## DESCRIPTION

`Vec_$$dpremult` multiplies the 4-element vector `start_vec` by the 4x4 matrix `matrix`.

In C, `vec_$$dpremult` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```
for (i = 0; i < 4; ++i) {
    result_vec[i] = 0.0;
    for (j = 0; j < 4; ++j)
        result_vec[i] += start_vec[j]
            * matrix[i][j];
}
```

In Pascal, `vec_$$dpremult` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```

for i := 1 to 4 do
  begin
    result_vec[i] := 0.0;
    for j := 1 to 4 do
      result_vec[i] := result_vec[i]
                      + start_vec[i]
                      * matrix[i,j];
    end
  end

```

In FORTRAN, `vec_$dpremult` applies *matrix* as a right transform to a row vector *start\_vec*, and the resulting operation is

```

do 10 i = 1, 4
  result_vec(i) = 0.0
  do 10 j = 1, 4
    result_vec(i) = result_vec(i)
                  + start_vec(j)
                  * matrix(j,i)
  10 continue

```

*start\_vec*

The vector to multiply by *matrix*.

*matrix* The matrix to multiply by *start\_vec*.

*result\_vec*

The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$dpostmult` transforms double-precision vectors from the other side.

#### SEE ALSO

`vec_$ipremult`, `vec_$ipremult16`, `vec_$premult`.

## NAME

`vec_$$dpremultn` – multiply a double-precision vector by a matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$dpremultn(
    double *start_vec,
    double *matrix,
    long int &m,
    long int &n,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$dpremultn(
    in start_vec: univ vec_$$double_vector;
    in matrix: univ vec_$$double_matrix;
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$$double_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
double precision start_vec(m), matrix(m, n), result_vec(n)
```

```
call vec_$$dpremultn(start_vec, matrix, m, n, result_vec)
```

## DESCRIPTION

`Vec_$$dpremultn` multiplies the  $m$ -element vector `start_vec` by the variably dimensioned matrix `matrix`, and supplies the resulting  $n$ -element vector in `result_vec`.

In C, `vec_$$dpremultn` applies the  $n \times m$  matrix `matrix` as a left transform to the  $m$ -element column vector `start_vec`, and supplies the transformed  $n$ -element result in `result_vec`:

```

for (i = 0; i < n; ++i) {
    result_vec[i] = 0.0;
    for (j = 0; j < m; ++j)
        result_vec[i] += start_vec[j]
                        * matrix[i][j];
}

```

In Pascal, `vec_$dpremultn` applies the  $n \times m$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to n do
begin
    result_vec[i] := 0.0;
    for j := 1 to m do
        result_vec[i] = result_vec[i]
                        + start_vec[j]
                        * matrix[i,j];
    end
end

```

In FORTRAN, `vec_$dpremultn` applies the  $m \times n$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, n
    result_vec(i) = 0.0
    do 10 j = 1, m
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(j,i)
    &
    &
10 continue

```

*start\_vec*

An  $m$ -element vector to multiply by *matrix*.

*matrix* A matrix to multiply *start\_vec* by.

*m* The number of elements in *start\_vec*.

*n* The number of elements in *result\_vec*.

*result\_vec*

An  $n$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$dpostmultn` transforms double-precision vectors from the other side.

#### SEE ALSO

`vec_$ipremultn`, `vec_$ipremultn16`, `vec_$premultn`.

## NAME

vec\_\$dsub – subtract double-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dsub(
    double *start_vec,
    double *sub_vec,
    long int &length,
    double *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dsub(
    in start_vec: univ vec_$double_vector;
    in sub_vec: univ vec_$double_vector;
    in length: integer32;
    out result_vec: univ vec_$double_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *start\_vec*(*nvec*), *sub\_vec*(*nvec*), *result\_vec*(*nvec*)  
integer\*4 *length*

call vec\_\$dsub(*start\_vec*, *sub\_vec*, *length*, *result\_vec*)

## DESCRIPTION

Vec\_\$dsub subtracts *length* elements of *sub\_vec* from *start\_vec* and supplies the difference in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] - sub_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    result_vec[i] := start_vec[i] - sub_vec[i];
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i) - sub_vec(i)
10 continue
```

*start\_vec*

The vector to subtract *sub\_vec* from.

*sub\_vec* The vector to subtract from *start\_vec*.

*length* The number of differences to calculate.

*result\_vec*

The difference of *start\_vec* and *sub\_vec*.

#### NOTES

When *vec\_\$dsub* is used to operate on matrixes in C and Pascal, *start\_vec*, *sub\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dsub\_i*, *vec\_\$isub*, *vec\_\$isub16*, *vec\_\$sub*.

**NAME**

`vec_$dsub_i` – subtract double-precision vectors in matrixes

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dsub_i(
    double *start_vec,
    long int &inc1,
    double *sub_vec,
    long int &inc2,
    long int &length,
    double *result_vec,
    long int &inc3)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dsub_i(
    in start_vec: univ vec_$double_vector;
    in inc1: integer32;
    in sub_vec: univ vec_$double_vector;
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_$double_vector;
    in inc3: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *start\_vec(nvec)*, *sub\_vec(nvec)*, *result\_vec(nvec)*  
integer\*4 *length*, *inc1*, *inc2*, *inc3*

```
call vec_$dsub_i(start_vec, inc1, sub_vec, inc2,
&                length, result_vec, inc3)
```

**DESCRIPTION**

`Vec_$dsub_i` subtracts *length* elements of *sub\_vec* selected by *inc2* from *length* elements of *start\_vec* selected by *inc1*, and supplies the result in the elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use `vec_$dsub_i` to subtract individual vectors in two matrixes and place the difference in a vector of another

matrix. To subtract the *N*th vector in matrix *Y* from the *M*th vector in matrix *X*, choose *inc2* equal to the number of vectors in matrix *Y* and *inc1* equal to the number of vectors in matrix *X*. Then place the *M*th element of matrix *X* at the beginning of *start\_vec*, and place the *N*th element of matrix *Y* at the beginning of *sub\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc3* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length, ++i) {
    result_vec[j] = start_vec[k] - sub_vec[l];
    j += inc3;
    k += inc1;
    l += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] - sub_vec[l];
    j := j + inc3;
    k := k + inc1;
    l := l + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
l = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) - sub_vec(l)
    j = j + inc3
    k = k + inc1
    l = l + inc2
10 continue

```

- start\_vec* The vector to subtract *sub\_vec* from.
- incl* An increment for the index of *start\_vec* that chooses which elements the elements of *sub\_vec* will be subtracted from.
- sub\_vec* The vector to subtract from *start\_vec*.
- inc2* An increment for the index of *sub\_vec* that chooses which elements will be subtracted from the elements of *start\_vec*.
- length* The number of scalar differences to calculate.
- result\_vec*  
The difference of *start\_vec* and *sub\_vec*.
- inc3* An increment for the index of *result\_vec* that chooses which elements will received the differences.

**NOTES**

In C and Pascal, `vec_$dsub_i` operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

`vec_$dsub`, `vec_$isub16_i`, `vec_$isub_i`, `vec_$sub_i`.

## NAME

`vec_$$sum` – sum the elements of a double-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
double vec_$$sum(
    double *vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$$sum(
    in vec: univ vec_$$double_vector;
    in length: integer32): double;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
double precision vec(nvec), sum
integer*4 length
```

```
sum = vec_$$sum(vec, length)
```

## DESCRIPTION

`Vec_$$sum` adds together *length* elements of the double-precision array *vector* and returns the sum.

In C, the resulting operation is

```
return_value = 0.0;
for (i = 0; i < length; ++i)
    return_value += vec[i];
```

In Pascal, the resulting operation is

```
return_value := 0.0;
for i := 1 to length do
    return_value := return_value + vec[i];
```

In FORTRAN, the resulting operation is

```
      vec_$dsum = 0.0  
      do 10 i = 1, length  
          vec_$dsum = vec_$dsum + vec(i)  
10    continue
```

*vec* The vector to sum.

*length* The number of elements in *vec* to sum.

**NOTES**

When *vec\_\$dsum* is used to operate on matrixes in C and Pascal, *vec* is a row vector; whereas in FORTRAN, it is a column vector.

**SEE ALSO**

*vec\_\$dsum\_i*, *vec\_\$isum*, *vec\_\$isum16*, *vec\_\$sum*.

## NAME

`vec_$$sum_i` – sum the elements of a vector in a double-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
double vec_$$sum_i(
    double *vec,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$$sum_i(
    in vec: univ vec_$$double_vector;
    in inc: integer32;
    in length: integer32): double;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
double precision vec(nvec), sum
integer*4 length, inc
```

```
sum = vec_$$sum_i(vec, inc, length)
```

## DESCRIPTION

`Vec_$$sum_i` adds *length* elements from the double-precision array *vector* selected by *inc* and returns the sum.

Through appropriate choice of *inc*, a program can use `vec_$$sum_i` to sum an individual vector in a matrix. To sum the *M*th vector in a matrix choose *inc* equal to the number of vectors in the matrix.

In C, the resulting operation is

```
return_value = 0.0;
j = 0;
for (i = 0; i < length; ++i) {
    return_value += vec[j];
    j += inc;
}
```

In Pascal, the resulting operation is

```
return_value := 0.0;
j := 1;
for i := 1 to length do
begin
    return_value := return_value + vec[j];
    j := j + inc;
end
```

In FORTRAN, the resulting operation is

```
vec_$dsum_i = 0.0
j = 1
do 10 i = 1, length
    vec_$dsum_i = vec_$dsum_i + vec(j)
    j = j + inc
10 continue
```

*vec* An array that contains the elements to sum.

*inc* An increment for the index of *vec* that selects which elements of *vec* are summed.

*length* The number of elements in *vec* to sum.

#### NOTES

In C and Pascal, *vec\_\$dsum\_i* sums a column vector; whereas in FORTRAN, it sums a row vector.

#### SEE ALSO

*vec\_\$dsum*, *vec\_\$isum16\_i*, *vec\_\$isum\_i*, *vec\_\$sum\_i*.

## NAME

`vec_$dswap` – swap two double-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dswap(
    double *vec1,
    double *vec2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dswap(
    var vec1: univ vec_$double_vector;
    var vec2: univ vec_$double_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
double precision vec1(nvec), vec2(nvec)
integer*4 length
```

```
call vec_$dswap(vec1, vec2, length)
```

## DESCRIPTION

`Vec_$dswap` swaps *length* elements between the double-precision vectors *vec1* and *vec2*.

In C, the resulting operation is

```
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
}
```

In Pascal, the resulting operation is

```
for i := 1 to length do
  begin
    temp := vec1[i];
    vec1[i] := vec2[i];
    vec2[i] := temp;
  end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
  temp = vec1(i)
  vec1(i) = vec2(i)
  vec2(i) = temp
10 continue
```

*vec1* The vector to be swapped with *vec2*.  
*vec2* The vector to be swapped with *vec1*.  
*length* The number of elements to swap.

#### NOTES

When `vec_$dswap` is used to operate on matrixes in C and Pascal, *vec1* and *vec2* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

`vec_$dswap_i`, `vec_$iswap`, `vec_$iswap16`, `vec_$swap`.

## NAME

vec\_\$dswap\_i – swap two vectors in a double-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dswap_i(
    double *vec1,
    long int &inc1,
    double *vec2,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dswap_i(
    var vec1: univ_vec_$double_vector;
    in inc1: integer32;
    var vec2: univ_vec_$double_vector;
    in inc2: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

double precision vec1(nvec), vec2(nvec)  
integer\*4 length, inc1, inc2

call vec\_\$dswap\_i(vec1, inc1, vec2, inc2, length)

## DESCRIPTION

Vec\_\$dswap\_i swaps the *length* elements of *vec1* selected by *inc1* with the *length* elements of *vec2* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use *vec\_\$dswap\_i* to swap vectors between two matrixes. To select the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vec1*. To swap the selected vector with the *N*th vector of the same or another matrix, choose *inc2* equal to the number of vectors in the same or other matrix, and place the *N*th element of the matrix array at the beginning of *vec2*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
    j += incl;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
    begin
        temp := vec1[i];
        vec1[i] := vec2[i];
        vec2[i] := temp;
        j := j + incl;
        k := k + inc2;
    end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          temp = vec1(j)
          vec1(j) = vec2(k)
          vec2(k) = temp
          j = j + incl
          k = k + inc2
10    continue

```

*vec1* The vector to be swapped with *vec2*.

*incl* The increment for the index of *vec1* that selects the elements to be swapped with *vec2*.

*vec2* The vector to be swapped with *vec1*.

*inc2* The increment for the index of *vec2* that selects the elements to be swapped with *vec1*.

VEC\_\$DSWAP\_I

VEC\_\$DSWAP\_I

*length* The number of elements to swap.

**NOTES**

In C and Pascal, `vec_$dswap_i` swaps column vectors; whereas in FORTRAN, it swaps row vectors.

**SEE ALSO**

`vec_$dswap`, `vec_$iswap16_i`, `vec_$iswap_i`, `vec_$swap_i`.

**NAME**

`vec_$dzero` – zero a double-precision vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dzero(
    double *vector,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include 'sys/ins/base.ins.pas';
%include 'sys/ins/vec.ins.pas';
```

```
procedure vec_$dzero(
    var vector: univ vec_$double_vector;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include 'sys/ins/base.ins.ftn'
%include 'sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *vector*(*nvec*)  
integer\*4 *length*

call `vec_$dzero`(*vector*, *length*)

**DESCRIPTION**

`Vec_$dzero` zeros the first *length* elements of the double-precision vector *vector*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = 0.0;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    vector[i] := 0.0;
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    vec(i) = 0.0
10 continue
```

*vector* The vector to be zeroed.

*length* The number of elements in *vector* to zero.

#### NOTES

In C and Pascal, `vec_$dzero` zeros row vectors; whereas in FORTRAN, it zeros column vectors.

#### SEE ALSO

`vec_$dzero_i`, `vec_$izero`, `vec_$izero16`, `vec_$zero`.

## NAME

`vec_$dzero_i` – zero a vector in a double-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$dzero_i(
    double *vector,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$dzero_i(
    var vector: univ vec_$double_vector;
    in inc: integer32;
    in length: integer32)
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *vector*(*nvec*)  
integer\*4 *length, inc*

call `vec_$dzero_i`(*vector, inc, length*)

## DESCRIPTION

`Vec_$dzero_i` zeros the *length* elements of the double-precision array *vector* selected by *inc*.

Through appropriate choice of *inc*, a program can use `vec_$dzero_i` to zero a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

j = 0;
for (i = 0; i < length; ++i) {
    vector[i] = 0.0;
    j += inc;
}

```

In Pascal, the resulting operation is

```

j := 1;
for i := 1 to length do
begin
    vector[i] := 0.0;
    j := j + inc;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      do 10 i = 1, length
          vec(j) = 0.0
          j = j + inc
10    continue

```

*vector* The vector to be zeroed.

*inc* An increment for the index of *vector* that chooses the elements to be zeroed.

*length* The number of elements in *vector* to zero.

#### NOTES

In C and Pascal, `vec_$dzero_i` zeros a column vector; whereas in FORTRAN, it zeros a row vector.

#### SEE ALSO

`vec_$dzero`, `vec_$zero16_i`, `vec_$zero_i`, `vec_$zero_i`.

**NAME**

`vec_$(i)add_constant` – add a scalar constant to a 32-bit integer vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(i)add_constant(
    long *start_vec,
    long int &length,
    long &constant,
    long *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(i)add_constant(
    in start_vec: univ vec_$(integer32)_vector;
    in length: integer32;
    in constant: integer32;
    out result_vec: univ vec_$(integer32)_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 256)

```
integer*4 start_vec(nvec), result_vec(nvec)
integer*4 constant
integer*4 length
```

call `vec_$(i)add_constant(start_vec, length, constant, result_vec)`

**DESCRIPTION**

`Vec_$(i)add_constant` adds the scalar *constant* to the 32-bit integer vector *start\_vec*, and supplies the result in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant + start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant + start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant + start_vec(i)
10 continue

```

*start\_vec*

The operand vector that *constant* will be added to.

*length* The number of elements in *start\_vec* that *constant* will be added to.

*constant*

A scalar constant to be added to *start\_vec*.

*result\_vec*

The vector resulting from adding *constant* to *start\_vec*.

#### NOTES

When *vec\_\$iadd\_constant* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_constant*, *vec\_\$add\_constant\_i*, *vec\_\$dadd\_constant*, *vec\_\$iadd\_constant16*.

## NAME

`vec_$$iadd_constant16` – add a scalar constant to a 16-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$$iadd_constant16(
    short *start_vec,
    long int &length,
    short &constant,
    short *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%iinclude '/sys/ins/vec.ins.pas';

procedure vec_$$iadd_constant16(
    in start_vec: univ vec_$$integer16_vector;
    in length: integer32;
    in constant: integer16;
    out result_vec: univ vec_$$integer16_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%iinclude '/sys/ins/vec.ins.ftn'

parameter (nvec = 256)

integer*4 start_vec(nvec), result_vec(nvec)
integer*4 constant
integer*4 length

call vec_$$iadd_constant16(start_vec, length, constant, result_vec)
```

## DESCRIPTION

`Vec_$$iadd_constant16` adds the scalar *constant* to the 16-bit integer vector *start\_vec*, and supplies the result in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant + start_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
  begin
    result_vec[i] := constant + start_vec[i];
  end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
  result_vec(i) = constant + start_vec(i)
10 continue
```

*start\_vec*

The operand vector that *constant* will be added to.

*length* The number of elements in *start\_vec* that *constant* will be added to.

*constant*

A scalar constant to be added to *start\_vec*.

*result\_vec*

The vector resulting from adding *constant* to *start\_vec*.

#### NOTES

When *vec\_\$iadd\_constant16* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_constant*, *vec\_\$add\_constant\_i*, *vec\_\$dadd\_constant*, *vec\_\$iadd\_constant*.

## NAME

`vec_$(iadd_constant16_i` – add a scalar to a vector in a 16-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(iadd_constant16_i(
    float *start_vec,
    long int &incl,
    long int &length,
    float &constant,
    float *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(iadd_constant16_i(
    in start_vec: univ vec_$(real_vector);
    in incl: integer32;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$(real_vector);
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *start\_vec*(*nvec*), *result\_vec*(*nvec*), *constant*  
integer\*4 *length*, *incl*, *inc2*

call `vec_$(iadd_constant16_i`(*start\_vec*, *incl*, *length*,  
& *constant*, *result\_vec*, *inc2*)

## DESCRIPTION

`vec_$(iadd_constant16_i` adds the scalar *constant* to elements of the 16-bit integer array *start\_vec* selected by *incl*, and puts the sums in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *incl* and *inc2*, a program can use `vec_$(iadd_constant16_i` to operate on individual vectors in a matrix. To add *constant* to the *M*th vector in a matrix, choose *incl* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the

result of the operation in the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant + start_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := constant + start_vec[j];
    k := k + inc2;
    j := j + incl;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
do 10 i = 1, length
    result_vec(k) = constant + start_vec(j)
    k = k + inc2
    j = j + incl
10 continue

```

*start\_vec*

The operand array whose elements will be summed with *constant*.

*incl* Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with *constant*.

*length* The number of elements in *start\_vec* that *constant* will be added to.

*constant*

The scalar value to be summed with elements of *start\_vec*.

*result\_vec*

The array resulting from summing *constant* with elements of *start\_vec*.

*inc2* Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums of *constant* with the elements of *start\_vec* selected by *inc1*.

**NOTES**

In C and Pascal, `vec_$iadd_constant16_i` operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

`vec_$add_constant`, `vec_$add_constant_i`, `vec_$dadd_constant_i`, `vec_$iadd_constant_i`.

## NAME

vec\_\$(i)add\_constant\_i – add a scalar to a vector in a 32-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(i)add_constant_i(
    float *start_vec,
    long int &inc1,
    long int &length,
    float &constant,
    float *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(i)add_constant_i(
    in start_vec: univ vec_$(real_vector);
    in inc1: integer32;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$(real_vector);
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

real start\_vec(nvec), result\_vec(nvec), constant  
integer\*4 length, inc1, inc2

call vec\_\$(i)add\_constant\_i(start\_vec, inc1, length,  
& constant, result\_vec, inc2)

## DESCRIPTION

Vec\_\$(i)add\_constant\_i adds the scalar *constant* to elements of the 32-bit integer array *start\_vec* selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use vec\_\$(i)add\_constant\_i to operate on individual vectors in a matrix. To add *constant* to the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the result of the

operation in the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant + start_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := constant + start_vec[j];
    k := k + inc2;
    j := j + incl;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result_vec(k) = constant + start_vec(j)
          k = k + inc2
          j = j + incl
10    continue

```

*start\_vec*

The operand array whose elements will be summed with *constant*.

*incl*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with *constant*.

*length*

The number of elements in *start\_vec* that *constant* will be added to.

*constant*

The scalar value to be summed with elements of *start\_vec*.

*result\_vec*

The array resulting from summing *constant* with elements of *start\_vec*.

*inc2* Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums of *constant* with the elements of *start\_vec* selected by *incl*.

## NOTES

In C and Pascal, *vec\_\$iadd\_constant\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

## SEE ALSO

*vec\_\$add\_constant*, *vec\_\$add\_constant\_i*, *vec\_\$dadd\_constant\_i*,  
*vec\_\$iadd\_constant16\_i*.

## NAME

`vec_$(i)add_vector` – add two 32-bit integer vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(i)add_vector(
    long *start_vec,
    long *add_vec,
    long int &length,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(i)add_vector(
    in start_vec: univ vec_$(integer32)_vector;
    in add_vec: univ vec_$(integer32)_vector;
    in length: integer32;
    out result_vec: univ vec_$(integer32)_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 start_vect(nvec), add_vect(nvec), result_vect(nvec)
integer*4 length
```

```
call vec_$(i)add_vector(start_vect, add_vect, length, result_vect)
```

## DESCRIPTION

`Vec_$(i)add_vector` adds the 32-bit integer vectors `start_vec` and `add_vec`, and supplies the vector sum in `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] + add_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := start_vec[i] + add_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

      do 10 i = 1, length
          result_vec(i) = start_vec(i) + add_vec(i)
10    continue

```

*start\_vec*

An addend vector.

*add\_vec*

An addend vector.

*length* Number of elements to sum. *Length* is usually just the order of the addends.

*result\_vec*

The vector that is the sum of *start\_vec* and *add\_vec*.

#### NOTES

When *vec\_\$iadd\_vector* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$add\_vector*, *vec\_\$dadd\_vector*, *vec\_\$iadd\_vector16*.

**NAME**

`vec_$(I)add_vector16` – add two 16-bit integer vectors

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(I)add_vector16(
    short *start_vec,
    short *add_vec,
    long int &length,
    short *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
#include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(I)add_vector16(
    in start_vec: univ vec_$(I)integer16_vector;
    in add_vec: univ vec_$(I)integer16_vector;
    in length: integer32;
    out result_vec: univ vec_$(I)integer16_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
#include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*2 start_vec(nvec), add_vec(nvec), result_vec(nvec)
integer*4 length
```

```
call vec_$(I)add_vector16(start_vec, add_vec, length, result_vec)
```

**DESCRIPTION**

`Vec_$(I)add_vector16` adds the single-precision vectors `start_vec` and `add_vec`, and supplies the vector sum in `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] + add_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    result_vec[i] := start_vec[i] + add_vec[i];
end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i) + add_vec(i)
10 continue
```

*start\_vec*     An addend vector.

*add\_vec*        An addend vector.

*length*        Number of elements to sum. *Length* is usually just the order of the addends.

*result\_vec*    The vector that is the sum of *start\_vec* and *add\_vec*.

#### NOTES

When `vec_$iadd_vector16` is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

`vec_$add_vector`, `vec_$dadd_vector`, `vec_$iadd_vector`.

## NAME

vec\_\$(I)add\_vector16\_i – add vectors in two 16-bit integer matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(I)add_vector16_i(
    short *start_vec,
    long int &inc1,
    short *add_vec,
    long int &inc2,
    long int &length,
    short *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(I)add_vector16_i(
    in start_vec: univ vec_$(I)integer16_vector;
    in inc1: integer32;
    in add_vec: univ vec_$(I)integer16_vector;
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_$(I)integer16_vector;
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

```
integer*2 start_vec(nvec), add_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3
```

```
call vec_$(I)add_vector16_i(start_vec, inc1, add_vec, inc2,
&                          length, result_vec, inc3)
```

## DESCRIPTION

Vec\_\$(I)add\_vector16\_i adds elements of the array *start\_vec*, selected by *inc1*, to elements of the array *add\_vec*, selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use vec\_\$(I)add\_vector16\_i to add individual vectors in two matrixes and place the sum in a

vector of another matrix. To add the *M*th vector in matrix *X* to the *N*th vector in matrix *Y*, choose *incl* equal to the number of vectors in matrix *X* and *inc2* equal to the number of vectors in matrix *Y*. Then place the *M*th element of matrix *X* at the beginning of *start\_vec*, and place the *N*th element of matrix *Y* at the beginning of *add\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
m = 0;
for (i = 0; i < length; ++i) {
    result_vec[j] = start_vec[k] + add_vec[m];
    j += inc3;
    k += incl1;
    m += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
m := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] + add_vec[m];
    j := j + inc3;
    k := k + incl1;
    m := m + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
m = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) + add_vec(m)
    j = j + inc3
    k = k + incl1
    m = m + inc2
10 continue

```

- start\_vec* An array whose elements will be summed with elements of *add\_vec*.
- inc1* Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with elements of *add\_vec*.
- add\_vec* An array whose elements will be summed with elements of *start\_vec*.
- inc2* Increment for the index of *add\_vec* used to select the elements of *add\_vec* that will be summed with elements of *start\_vec*.
- length* The number of elements in *start\_vec* that will be summed with elements of *add\_vec*.
- result\_vec* An array to contain the *length* sums of elements of *start\_vec* and *add\_vec*.
- inc3* Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums.

**NOTES**

In C and Pascal, `vec_$iadd_vector16_i` operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

`vec_$add_vector`, `vec_$dadd_vector_i`, `vec_$iadd_vector_i`.

## NAME

vec\_\$(I)add\_vector\_i – add vectors in two 32-bit integer matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(I)add_vector_i(
    long *start_vec,
    long int &inc1,
    long *add_vec,
    long int &inc2,
    long int &length,
    long *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(I)add_vector_i(
    in start_vec: univ vec_$(I)integer32_vector;
    in inc1: integer32;
    in add_vec: univ vec_$(I)integer32_vector;
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_$(I)integer32_vector;
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

```
integer*4 start_vec(nvec), add_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3
```

```
call vec_$(I)add_vector_i(start_vec, inc1, add_vec, inc2,
& length, result_vec, inc3)
```

## DESCRIPTION

Vec\_\$(I)add\_vector\_i adds elements of the array *start\_vec*, selected by *inc1*, to elements of the array *add\_vec*, selected by *inc1*, and puts the sums in elements of the array *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use *vec\_\$(I)add\_vector\_i* to add individual vectors in two matrixes and place the sum in a

vector of another matrix. To add the *M*th vector in matrix *X* to the *N*th vector in matrix *Y*, choose *inc1* equal to the number of vectors in matrix *X* and *inc2* equal to the number of vectors in matrix *Y*. Then place the *M*th element of matrix *X* at the beginning of *start\_vec*, and place the *N*th element of matrix *Y* at the beginning of *add\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
m = 0;
for (i = 0; i < length; ++i) {
    result_vec[j] = start_vec[k] + add_vec[m];
    j += inc3;
    k += inc1;
    m += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
m := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] + add_vec[m];
    j := j + inc3;
    k := k + inc1;
    m := m + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
m = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) + add_vec(m)
    j = j + inc3
    k = k + inc1
    m = m + inc2
10 continue

```

- start\_vec* An array whose elements will be summed with elements of *add\_vec*.
- inc1* Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be summed with elements of *add\_vec*.
- add\_vec* An array whose elements will be summed with elements of *start\_vec*.
- inc2* Increment for the index of *add\_vec* used to select the elements of *add\_vec* that will be summed with elements of *start\_vec*.
- length* The number of elements in *start\_vec* that will be summed with elements of *add\_vec*.
- result\_vec* An array to contain the *length* sums of elements of *start\_vec* and *add\_vec*.
- inc3* Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the sums.

**NOTES**

In C and Pascal, *vec\_\$iadd\_vector\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$add\_vector\_i*, *vec\_\$dadd\_vector\_i*, *vec\_\$iadd\_vector16\_i*.

**NAME**

`vec_$icopy` – copy a 32-bit integer vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$icopy(
    long *start_vec,
    long *result_vec,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$icopy(
    in start_vec: univ vec_$integer32_vector;
    out result_vec: univ vec_$integer32_vector;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 start_vec(nvec), result_vec(nvec)
integer*4 length
```

```
call vec_$icopy(start_vec, result_vec, length)
```

**DESCRIPTION**

`Vec_$icopy` copies *length* elements from *start\_vec* to *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    result_vec[i] := start_vec[i];
end
```

In FORTRAN, the resulting operation is

```
      do 10 i = 1, length
         result_vec(i) = start_vec(i)
10    continue
```

*start\_vec*

The vector that *result\_vec* will be copied from.

*result\_vec*

The vector that *start\_vec* will be copied to.

*length*

The number of elements to copy from *start\_vec* to *result\_vec*.

#### NOTES

When *vec\_\$icopy* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$copy*, *vec\_\$dcopy*, *vec\_\$icopy\_i*, *vec\_\$icopy16*.

## NAME

`vec_$icopy16` – copy a 16-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$icopy16(
    short *start_vec,
    short *result_vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$icopy16(
    in start_vec: univ vec_$integer16_vector;
    out result_vec: univ vec_$integer16_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 start_vec(nvec), result_vec(nvec)
integer*4 length
```

```
call vec_$icopy16(start_vec, result_vec, length)
```

## DESCRIPTION

`Vec_$icopy16` copies *length* elements from *start\_vec* to *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    result_vec[i] := start_vec[i];
end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i)
10 continue
```

*start\_vec*

The vector that *result\_vec* will be copied from.

*result\_vec*

The vector that *start\_vec* will be copied to.

*length*

The number of elements to copy from *start\_vec* to *result\_vec*.

#### NOTES

When *vec\_\$copy16* is used to operate on matrixes in C and Pascal, *start\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$copy*, *vec\_\$dcopy*, *vec\_\$copy*, *vec\_\$copy16\_i*.

**NAME**

**vec\_\$icopy16\_i** – copy a vector from one 16-bit integer matrix to another

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$icopy16_i(
    short *start_vec,
    long int &inc1,
    short *result_vec,
    long int &inc2,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$icopy16_i(
    in start_vec: univ vec_$integer16_vector;
    in inc1: integer32;
    out result_vec: univ vec_$integer16_vector;
    in inc2: integer32;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*2 start_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2
```

```
call vec_$icopy16_i(start_vec, inc1, result_vec, inc2, length)
```

**DESCRIPTION**

**Vec\_\$icopy16\_i** copies *length* elements of the 16-bit integer array *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use **vec\_\$icopy16\_i** to copy a vector from one matrix to another. To copy the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the copy into the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = start_vec[j];
    k += inc2;
    j += inc1;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := start_vec[j];
    k := k + inc2;
    j := j + inc1;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result_vec(j) = start_vec(k)
          j = j + inc2
          k = k + inc1
10    continue

```

*start\_vec*

The array whose elements will be copied to *result\_vec*.

*inc1*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be copied to *result\_vec*.

*result\_vec*

The array resulting from copying elements of *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

*inc2*

Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the copies of the elements of *start\_vec* selected by *inc1*.

*length*

The number of elements in *start\_vec* that will be copied to *result\_vec*.

**NOTES**

In C and Pascal, `vec_$icopy16_i` copies column vectors; whereas in FORTRAN, it copies row vectors.

**SEE ALSO**

`vec_$copy_i`, `vec_$dcopy_i`, `vec_$icopy16`, `vec_$icopy_i`.

## NAME

`vec_$icopy_i` – copy a vector from one 32-bit integer matrix to another

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$icopy_i(
    long *start_vec,
    long int &incl,
    long *result_vec,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$icopy_i(
    in start_vec: univ vec_$integer32_vector;
    in incl: integer32;
    out result_vec: univ vec_$integer32_vector;
    in inc2: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%cinclude '/sys/ins/base.ins.ftn'
%cinclude '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*4 start_vec(nvec), result_vec(nvec)
integer*4 length, incl, inc2
```

call `vec_$icopy_i(start_vec, incl, result_vec, inc2, length)`

## DESCRIPTION

`Vec_$icopy_i` copies *length* elements of the 32-bit integer array *start\_vec* selected by *incl* into elements of *result\_vec* selected by *inc2*.

Through appropriate choice of *incl* and *inc2*, a program can use `vec_$icopy_i` to copy a vector from one matrix to another. To copy the *M*th vector in a matrix, choose *incl* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *start\_vec*. To place the copy into the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = start_vec[j];
    k += inc2;
    j += inc1;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := start_vec[j];
    k := k + inc2;
    j := j + inc1;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k)
    j = j + inc2
    k = k + inc1
10 continue

```

*start\_vec*

The array whose elements will be copied to *result\_vec*.

*inc1*

Increment for the index of *start\_vec* used to select the elements of *start\_vec* that will be copied to *result\_vec*.

*result\_vec*

The array resulting from copying elements of *start\_vec* selected by *inc1* into elements of *result\_vec* selected by *inc2*.

*inc2*

Increment for the index of *result\_vec* used to select the elements of *result\_vec* that will receive the copies of the elements of *start\_vec* selected by *inc1*.

*length*

The number of elements in *start\_vec* that will be copied to *result\_vec*.

VEC\_\$ICOPY\_I

VEC\_\$ICOPY\_I

**NOTES**

In C and Pascal, `vec_$icopy_i` copies column vectors; whereas in FORTRAN, it copies row vectors.

**SEE ALSO**

`vec_$copy_i`, `vec_$dcopy_i`, `vec_$icopy`, `vec_$icopy16_i`.

## NAME

`vec_$idot` – return the dot product of two 32-bit integer vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
long vec_$idot(
    long *vector1,
    long *vector2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$idot(
    in vector1: univ vec_$integer32_vector;
    in vector2: univ vec_$integer32_vector;
    in length: integer32): integer32;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 vector1(nvec), vector2(nvec), result
integer*4 length
```

```
result = vec_$idot(vector1, vector2, length)
```

## DESCRIPTION

`Vec_$idot` returns the dot (scalar) product of two single-precision vectors, `vector1` and `vector2`.

In C, the resulting operation is

```
return_value = 0;
for (i = 0; i < length; ++i)
    return_value += vector1[i] * vector2[i];
```

In Pascal, the resulting operation is

```

return_value := 0;
for i := 1 to length do
begin
return_value := return_value
                + vector1[i] * vector2[i];
end

```

In FORTRAN, the resulting operation is

```

vec_$idot = 0
do 10 i = 1,length
    vec_$idot = vec_$idot
    &          + vector1(i) * vector2(i)
10 continue

```

*vector1* A vector.

*vector2* Another vector.

*length* The number of elements to use in calculating the dot (scalar) product.

#### NOTES

When *vec\_\$idot* is used on matrixes in C or Pascal, *vector1* and *vector2* are row vectors; whereas in FORTRAN they are column vectors.

#### SEE ALSO

*vec\_\$ddot*, *vec\_\$dot*, *vec\_\$idot16*, *vec\_\$idot\_i*.

**NAME**

**vec\_\$\_idot16** – return the dot product of two 16-bit integer vectors

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
short vec_$_idot16(
    short *vector1,
    short *vector2,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$_idot16(
    in vector1: univ vec_$_integer16_vector;
    in vector2: univ vec_$_integer16_vector;
    in length: integer32): integer16;
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*2 vector1(nvec), vector2(nvec), result
integer*4 length
```

```
result = vec_$_idot16(vector1, vector2, length)
```

**DESCRIPTION**

**Vec\_\$\_idot16** returns the dot (scalar) product of two single-precision vectors, *vector1* and *vector2*.

In C, the resulting operation is

```
return_value = 0;
for (i = 0; i < length; ++i)
    return_value += vector1[i] * vector2[i];
```

In Pascal, the resulting operation is

```

return_value := 0;
for i := 1 to length do
begin
  return_value := return_value
                + vector1[i] * vector2[i];
end

```

In FORTRAN, the resulting operation is

```

vec_$idot16 = 0
do 10 i = 1,length
  vec_$idot16 = vec_$idot16
&               + vector1(i) * vector2(i)
10 continue

```

*vector1* A vector.

*vector2* Another vector.

*length* The number of elements to use in calculating the dot (scalar) product.

#### NOTES

When *vec\_\$idot16* is used on matrixes in C or Pascal, *vector1* and *vector2* are row vectors; whereas in FORTRAN they are column vectors.

#### SEE ALSO

*vec\_\$ddot*, *vec\_\$dot*, *vec\_\$idot*, *vec\_\$idot16\_i*.

## NAME

`vec_${IDOT16}_i` – return the dot product of two vectors in 16-bit integer matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
short vec_${IDOT16}_i(
    short *vector1,
    long int &inc1,
    short *vector2,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_${IDOT16}_i(
    in vector1: univ vec_${INTEGER16}_vector;
    in inc1: integer32;
    in vector2: univ vec_${INTEGER16}_vector;
    in inc2: integer32;
    in length: integer32): integer16;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*2 *vector1*(*nvec*), *vector2*(*nvec*), *result*  
integer\*4 *length*, *inc1*, *inc2*

*result* = `vec_${IDOT16}_i`(*vector1*, *inc1*, *vector2*, *inc2*, *length*)

## DESCRIPTION

`Vec_${IDOT16}_i` returns the dot (scalar) product of two vectors from the 16-bit integer arrays *vector1* and *vector2*.

In C, the resulting operation is

```

j = 0;
k = 0;
return_value = 0;
for (i = 0; i < length; ++i) {
    return_value += vector1[j] * vector2[k];
    j += inc1;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
return_value := 0;
for i := 1 to length do
begin
    return_value := return_value
                    + vector1[j] * vector2[k];
    j := j + inc1;
    k := k + inc2;
end;

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      vec_$idot16_i = 0
      do 10 i = 1,length
          vec_$idot16_i = vec_$idot16_i
&                + vector1(j) * vector2(k)
                j = j + inc1
                k = k + inc2
10  continue

```

*vector1* An array containing one of the vectors to use in calculating the dot product.

*inc1* An increment for *vector1* that chooses which elements will be used to calculate the product.

*vector2* An array containing the other vector to use in calculating the dot product.

*inc2* An increment for *vector2* that chooses which elements will be used to calculate the product.

*length* The number of elements from *vector1* or *vector2* to use in calculating the dot product.

**NOTES**

When `vec_$idot16_i` is used on matrixes in C or Pascal, *vector1* and *vector2* are column vectors; whereas in FORTRAN they are row vectors.

**SEE ALSO**

`vec_$ddot_i`, `vec_$dot_i`, `vec_$idot16`, `vec_$idot_i`.

**NAME**

`vec_$idot_i` – return the dot product of two vectors in 32-bit integer matrices

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
long vec_$idot_i(
    long *vector1,
    long int &inc1,
    long *vector2,
    long int &inc2,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$idot_i(
    in vector1: univ vec_$integer32_vector;
    in inc1: integer32;
    in vector2: univ vec_$integer32_vector;
    in inc2: integer32;
    in length: integer32): integer32;
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

**parameter** (*nvec* = 10)

**integer\*4** *vector1*(*nvec*), *vector2*(*nvec*), *result*  
**integer\*4** *length*, *inc1*, *inc2*

*result* = `vec_$idot_i`(*vector1*, *inc1*, *vector2*, *inc2*, *length*)

**DESCRIPTION**

`Vec_$idot_i` returns the dot (scalar) product of two vectors from the 32-bit integer arrays *vector1* and *vector2*.

In C, the resulting operation is

```

j = 0;
k = 0;
return_value = 0;
for (i = 0; i < length; ++i) {
    return_value += vector1[j] * vector2[k];
    j += inc1;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
return_value := 0;
for i := 1 to length do
    begin
        return_value := return_value
            + vector1[j] * vector2[k];
        j := j + inc1;
        k := k + inc2;
    end;

```

In FORTRAN,  
the resulting operation is

```

      j = 1
      k = 1
      vec_$idot_i = 0
      do 10 i = 1,length
          vec_$idot_i = vec_$idot_i
&                + vector1(j) * vector2(k)
                j = j + inc1
                k = k + inc2
10  continue

```

*vector1* An array containing one of the vectors to use in calculating the dot product.

*inc1* An increment for *vector1* that chooses which elements will be used to calculate the product.

*vector2* An array containing the other vector to use in calculating the dot product.

*inc2* An increment for *vector2* that chooses which elements will be used to calculate the product.

VEC\_\$IDOT\_I

VEC\_\$IDOT\_I

*length* The number of elements from *vector1* or *vector2* to use in calculating the dot product.

NOTES

When `vec_$idot_i` is used on matrixes in C or Pascal, *vector1* and *vector2* are column vectors; whereas in FORTRAN they are row vectors.

SEE ALSO

`vec_$ddot_i`, `vec_$dot_i`, `vec_$idot`, `vec_$idot16_i`.

## NAME

`vec_$$iinit` – initialize a 32-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$iinit(
    long *vector,
    long int &length,
    long &constant)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$iinit(
    var vector: univ vec_$$integer32_vector;
    in length: integer32;
    in constant: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*4 vector(nvec), constant
integer*4 length
```

call `vec_$$iinit(vector, length, constant)`

## DESCRIPTION

`Vec_$$iinit` sets *length* elements of *vector* to the 32-bit integer value *constant*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = constant;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    vector[i] := constant;
end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    vector(i) = constant
10 continue
```

*vector* The vector to initialize.

*length* The number of elements in *vector* to initialize.

*constant*

The value that the elements of *vector* should be set to.

#### NOTES

In C and Pascal, `vec_$iinit` initializes a row vector; whereas in FORTRAN, it initializes a column vector.

#### SEE ALSO

`vec_$dinit`, `vec_$iinit16`, `vec_$iinit_i`, `vec_$init`.

## NAME

`vec_$$init16` – initialize a 16-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$init16(
    short *vector,
    long int &length,
    short &constant)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$init16(
    var vector: univ vec_$$integer16_vector;
    in length: integer32;
    in constant: integer16);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*2 *vector*(*nvec*), *constant*  
integer\*4 *length*

call `vec_$$init16`(*vector*, *length*, *constant*)

## DESCRIPTION

`Vec_$$init16` sets *length* elements of *vector* to the 16-bit integer value *constant*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = constant;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    begin
        vector[i] := constant;
    end .
```

In FORTRAN, the resulting operation is

```
      do 10 i = 1, length
         vector(i) = constant
10    continue
```

*vector* The vector to initialize.

*length* The number of elements in *vector* to initialize.

*constant*

The value that the elements of *vector* should be set to.

#### NOTES

In C and Pascal, `vec_$iinit16` initializes a row vector; whereas in FORTRAN, it initializes a column vector.

#### SEE ALSO

`vec_$dinit`, `vec_$iinit16`, `vec_$iinit_i`, `vec_$init`.

**NAME**

`vec_$imat_mult` – multiply two 4x4 32-bit integer matrixes

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$imat_mult(
    long *matrix1,
    long *matrix2,
    long *out_matrix)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imat_mult(
    in matrix1: univ vec_$integer32_matrix;
    in matrix2: univ vec_$integer32_matrix;
    out out_matrix: univ vec_$integer32_matrix);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 matrix1(4, 4), matrix2(4, 4), out_matrix(4, 4)
```

```
call vec_$imat_mult(matrix1, matrix2, out_matrix)
```

**DESCRIPTION**

`Vec_$imat_mult` multiplies two 4x4 matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, `vec_$imat_mult` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```
for (i = 0; i < 4; ++i)
    for (j = 0; j < 4; ++j) {
        out_mat[j,i] = 0;
        for (k = 0; k < 4; ++k)
            out_mat[j][i] += matrix1[k][i]
                * matrix2[j][k];
    }
```

In Pascal, `vec_$imat_mult` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to 4 do
  for j := 1 to 4 do
    begin
      out_mat[j,i] := 0;
      for k := 1 to 4 do
        out_mat[j,i] := out_mat[j,i]
          + matrix1[k,i]
          * matrix2[j,k];
      end;
    end;
  end;
end;

```

In FORTRAN, `vec_$imat_mult` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, 4
  do 10 j = 1, 4
    out_mat(i,j) = 0
    do 10 k = 1, 4
      out_mat(i,j) = out_mat(i,j)
        + matrix1(i,k)
        * matrix2(k,j)
    &
  &
10 continue

```

*matrix1* A 4x4 matrix.  
*matrix2* Another 4x4 matrix.  
*out\_matrix*  
 The product of *matrix1* and *matrix2*.

#### NOTES

`Vec_$imat_multn` performs the same operations for variably dimensioned matrixes.

#### SEE ALSO

`vec_$dmat_mult`, `vec_$imat_mult16`, `vec_$mat_mult`.

## NAME

vec\_\$imat\_mult16 – multiply two 4x4 16-bit integer matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$imat_mult16(
    short *matrix1,
    short *matrix2,
    short *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imat_mult16(
    in matrix1: univ vec_$integer16_matrix;
    in matrix2: univ vec_$integer16_matrix;
    out out_matrix: univ vec_$integer16_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*2 matrix1(4, 4), matrix2(4, 4), out_matrix(4, 4)
```

```
call vec_$imat_mult16(matrix1, matrix2, out_matrix)
```

## DESCRIPTION

Vec\_\$imat\_mult16 multiplies two 4x4 matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, vec\_\$imat\_mult16 calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```
for (i = 0; i < 4; ++i)
    for (j = 0; j < 4; ++j) {
        out_mat[j,i] = 0;
        for (k = 0; k < 4; ++k)
            out_mat[j][i] += matrix1[k][i]
                * matrix2[j][k];
    }
```

In Pascal, vec\_\$imat\_mult16 calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to 4 do
  for j := 1 to 4 do
    begin
      out_mat[j,i] := 0;
      for k := 1 to 4 do
        out_mat[j,i] := out_mat[j,i]
          + matrix1[k,i]
          * matrix2[j,k];
      end;
    end;
  end;
end;

```

In FORTRAN, `vec_$imat_mult16` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, 4
  do 10 j = 1, 4
    out_mat(i,j) = 0
    do 10 k = 1, 4
      out_mat(i,j) = out_mat(i,j)
        + matrix1(i,k)
        * matrix2(k,j)
    &
  &
10 continue

```

*matrix1* A 4×4 matrix.  
*matrix2* Another 4×4 matrix.  
*out\_matrix*  
 The product of *matrix1* and *matrix2*.

#### NOTES

`Vec_$imat_multn16` performs the same operations for variably dimensioned matrices.

#### SEE ALSO

`vec_$dmat_mult`, `vec_$imat_mult`, `vec_$mat_mult`.

## NAME

vec\_\$imat\_multn – multiply two 32-bit integer matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
```

```
#include <apollo/vec.h>
```

```
void vec_$imat_multn(
    long *matrix1,
    long *matrix2,
    long int &m,
    long int &n,
    long int &s,
    long *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
```

```
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imat_multn(
    in matrix1: univ vec_$integer32_matrix;
    in matrix2: univ vec_$integer32_matrix;
    in m: integer32;
    in n: integer32;
    in s: integer32;
    out out_matrix: univ vec_$integer32_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
```

```
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n, s
parameter (m = 3, n = 4, s = 5)
```

```
integer*4 matrix1(m, n), matrix2(n, s), out_matrix(m, s)
```

```
call vec_$imat_multn(matrix1, matrix2, m, n, s, out_matrix)
```

## DESCRIPTION

Vec\_\$imat\_multn multiplies two variably dimensioned matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, vec\_\$imat\_multn calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for (i = 0; i < m; ++i)
  for (j = 0; j < s; ++j) {
    out_matrix[j][i] = 0;
    for (k = 0; k < n; ++k)
      out_matrix[j][i] += matrix1[k][i]
                        * matrix2[j][k];
  }

```

In Pascal, `vec_$imat_multn` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to m do
  for j := 1 to s do
    begin
      out_matrix[j,i] = 0;
      for k := 1 to n do
        out_matrix[j,i] := out_matrix[j,i]
                          + matrix1[k,i]
                          * matrix2[j,k];
      end;
    end;

```

In FORTRAN, `vec_$imat_multn` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, m
  do 10 j = 1, s
    out_matrix(i,j) = 0
    do 10 k = 1, n
      out_matrix(i,j) = out_matrix(i,j)
                        + matrix1(i,k)
                        * matrix2(k,j)
    &
    &
  10 continue

```

*matrix1* A matrix to be multiplied.

*matrix2* Another matrix to be multiplied.

*m, n, s* The various matrix dimensions.

*out\_matrix*

The product of *matrix1* and *matrix2*.

#### SEE ALSO

`vec_$dmat_multn`, `vec_$imat_multn`, `vec_$mat_multn`.

## NAME

vec\_\$imat\_multn16 – multiply two 16-bit integer matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$imat_multn16(
    short *matrix1,
    short *matrix2,
    long int &m,
    long int &n,
    long int &s,
    short *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$imat_multn16(
    in matrix1: univ vec_$integer16_matrix;
    in matrix2: univ vec_$integer16_matrix;
    in m: integer32;
    in n: integer32;
    in s: integer32;
    out out_matrix: univ vec_$integer16_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

integer*2 m, n, s
parameter (m = 3, n = 4, s = 5)

integer*4 matrix1(m, n), matrix2(n, s), out_matrix(m, s)

call vec_$imat_multn16(matrix1, matrix2, m, n, s, out_matrix)
```

## DESCRIPTION

Vec\_\$imat\_multn16 multiplies two variably dimensioned matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, vec\_\$imat\_multn16 calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for (i = 0; i < m; ++i)
  for (j = 0; j < s; ++j) {
    out_matrix[j][i] = 0;
    for (k = 0; k < n; ++k)
      out_matrix[j][i] += matrix1[k][i]
                        * matrix2[j][k];
  }

```

In Pascal, `vec_$imat_multn16` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to m do
  for j := 1 to s do
    begin
      out_matrix[j,i] = 0;
      for k := 1 to n do
        out_matrix[j,i] := out_matrix[j,i]
                          + matrix1[k,i]
                          * matrix2[j,k];
      end;
    end;

```

In FORTRAN, `vec_$imat_multn16` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, m
  do 10 j = 1, s
    out_matrix(i,j) = 0
    do 10 k = 1, n
      out_matrix(i,j) = out_matrix(i,j)
                      + matrix1(i,k)
                      * matrix2(k,j)
    &
  &
10 continue

```

*matrix1* A matrix to be multiplied.

*matrix2* Another matrix to be multiplied.

*m, n, s* The various matrix dimensions.

*out\_matrix*

The product of *matrix1* and *matrix2*.

#### SEE ALSO

`vec_$dmat_multn`, `vec_$imat_multn`, `vec_$mat_multn`.

**NAME**

`vec_$imax` – find the maximum absolute value in a 32-bit integer vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$imax(
    long *vector,
    long int &length,
    long *result,
    long int *location)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imax(
    in vector: univ vec_$integer32_vector;
    in length: integer32;
    out result: integer32;
    out location: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*4 vector(nvec), result
integer*4 length, location
```

call `vec_$imax(vector, length, result, location)`

**DESCRIPTION**

`Vec_$imax` searches through *length* elements of *vector* and supplies the value and location of the element with the greatest absolute value.

In C, the resulting operation is

```

result = abs(vector[0]);
location = 1;
for (i = 1; i < length; ++i)
    if (abs(vector[i]) > result) {
        location = i + 1;
        result = abs(vector[i]);
    }

```

In Pascal, the resulting operation is

```

result := abs(vector[1]);
location = 1;
for 10 i := 2 to length do
    if (abs(vector[i]) > result) then
        begin
            location := i;
            result := abs(vector[i]);
        end
end

```

In FORTRAN, the resulting operation is

```

result = iabs(vector(1))
location = 1
do 10 i = 2, length
    if (iabs(vector(i)) .gt. result) then
        location = i
        result = iabs(vector(i))
    endif
10 continue

```

*vector* The vector to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, *vec\_\$imax* searches a row vector; whereas in FORTRAN, it searches a column vector.

VEC\_\$IMAX

VEC\_\$IMAX

SEE ALSO

vec\_\$dmax, vec\_\$imax\_i, vec\_\$imax16, vec\_\$max.

## NAME

`vec_$imax16` – find the maximum absolute value in a 16-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$imax16(
    short *vector,
    long int &length,
    short *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imax16(
    in vector: univ vec_$integer16_vector;
    in length: integer32;
    out result: integer16;
    out location: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*2 *vector*(*nvec*), *result*  
integer\*4 *length*, *location*

call `vec_$imax16`(*vector*, *length*, *result*, *location*)

## DESCRIPTION

`Vec $imax16` searches through *length* elements of *vector* and supplies the value and location of the element with the greatest absolute value.

In C, the resulting operation is

```
result = (short)abs(vector[0]);
location = 1;
for (i = 1; i < length; ++i)
    if ((short)abs(vector[i]) > result) {
        location = i + 1;
        result = (short)abs(vector[i]);
    }
```

In Pascal, the resulting operation is

```
result := abs(vector[1]);
location = 1;
for 10 i := 2 to length do
    if (abs(vector[i]) > result) then
        begin
            location := i;
            result := abs(vector[i]);
        end
```

In FORTRAN, the resulting operation is

```
result = iabs(vector(1))
location = 1
do 10 i = 2, length
    if (iabs(vector(i)) .gt. result) then
        location = i
        result = iabs(vector(i))
    endif
10 continue
```

*vector* The vector to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, `vec_$imax16` searches a row vector; whereas in FORTRAN, it searches a column vector.

VEC\_\$IMAX16

VEC\_\$IMAX16

SEE ALSO

vec\_\$dmax, vec\_\$imax, vec\_\$imax16\_i, vec\_\$max.

## NAME

`vec_$imax16_i` – find the maximum absolute value in a vector from a 16-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$imax16_i(
    short *vector,
    long int &inc,
    long int &length,
    short *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$imax16_i(
    in vector: univ vec_$integer16_vector;
    in inc: integer32;
    in length: integer32;
    out result: integer16;
    out location: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

integer*2 vector(nvec), result
integer*4 length, inc, location

call vec_$imax16_i(vector, inc, length, result, location)
```

## DESCRIPTION

`Vec_$imax16_i` searches through the *length* elements of *vector* selected by *inc*, and supplies the value and location of the element with the greatest absolute value.

Through appropriate choice of *inc*, a program can use `vec_$imax16_i` to search a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

result = (short)abs(vector[0]);
location = 1;
j = inc;
for (i = 1; i < length, ++i) {
    if ((short)abs(vector[j]) > result) {
        location = i + 1;
        result = (short)abs(vector[j]);
    }
    j += inc;
}

```

In Pascal, the resulting operation is

```

result := abs(vector[1]);
location := 1;
j := 1 + inc;
for 10 i := 2 to length do
    begin
        if (abs(vector[j]) > result) then
            begin
                location := i;
                result := abs(vector[j]);
            end
        j := j + inc;
    end

```

In FORTRAN, the resulting operation is

```

result = iabs(vector(1))
location = 1
j = 1 + inc
do 10 i = 2, length
    if (iabs(vector(j)) .gt. result) then
        location = i
        result = iabs(vector(j))
    endif
    j = j + inc
10 continue

```

*vector* The array to search.

*inc* An increment for the index of *vector* that selects the elements to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, `vec_$imax16_i` searches a column vector; whereas in FORTRAN, it searches a row vector.

#### SEE ALSO

`vec_$dmax_i`, `vec_$imax16`, `vec_$imax_i`, `vec_$max_i`.

## NAME

vec\_\$imax\_i – find the maximum absolute value in a vector from a 32-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$imax_i(
    long *vector,
    long int &inc,
    long int &length,
    long *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
#include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imax_i(
    in vector: univ vec_$integer32_vector;
    in inc: integer32;
    in length: integer32;
    out result: integer32;
    out location: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
#include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*4 *vector*(*nvec*), *result*  
integer\*4 *length*, *inc*, *location*

call vec\_\$imax\_i(*vector*, *inc*, *length*, *result*, *location*)

## DESCRIPTION

Vec\_\$imax\_i searches through the *length* elements of *vector* selected by *inc*, and supplies the value and location of the element with the greatest absolute value.

Through appropriate choice of *inc*, a program can use vec\_\$imax\_i to search a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

result = abs(vector[0]);
location = 1;
j = inc;
for (i = 1; i < length, ++i) {
    if (abs(vector[j]) > result) {
        location = i + 1;
        result = abs(vector[j]);
    }
    j += inc;
}

```

In Pascal, the resulting operation is

```

result := abs(vector[1]);
location := 1;
j := 1 + inc;
for 10 i := 2 to length do
    begin
        if (abs(vector[j]) > result) then
            begin
                location := i;
                result := abs(vector[j]);
            end
        j := j + inc;
    end

```

In FORTRAN, the resulting operation is

```

        result = iabs(vector(1))
        location = 1
        j = 1 + inc
        do 10 i = 2, length
            if (iabs(vector(j)) .gt. result) then
                location = i
                result = iabs(vector(j))
            endif
            j = j + inc
10    continue

```

*vector* The array to search.

*inc* An increment for the index of *vector* that selects the elements to search.

- length* The number of elements to search.
- result* The maximum absolute value of all the elements searched.
- location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

**NOTES**

In C and Pascal, `vec_$imax_i` searches a column vector; whereas in FORTRAN, it searches a row vector.

**SEE ALSO**

`vec_$dmax_i`, `vec_$imax`, `vec_$imax16_i`, `vec_$max_i`.

## NAME

`vec_$simult_add` – scale and add one 32-bit integer vector to another

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$simult_add(
    long *add_vec,
    long *mult_vec,
    long int &length,
    long &constant,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$simult_add(
    in add_vec: univ vec_$integer32_vector;
    in mult_vec: univ vec_$integer32_vector;
    in length: integer32;
    in constant: integer32;
    out result_vec: univ vec_$integer32_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 add_vec(nvec), mult_vec(nvec), result_vec(nvec), constant
integer*4 length
```

```
call vec_$simult_add(add_vec, mult_vec, length,
& constant, result_vec)
```

## DESCRIPTION

`Vec_$simult_add` multiplies the vector `mult_vec` by the scalar `constant`, adds the product to the vector `add_vec`, and supplies the resulting vector in `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = add_vec[i]
        + constant * mult_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    result_vec[i] := add_vec[i]
                  + constant * mult_vec[i];
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = add_vec(i)
&                + constant * mult_vec(i)
10 continue
```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*length* The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

When *vec\_\$simult\_add* is used to operate on matrixes in C and Pascal, *add\_vec*, *mult\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_add*, *vec\_\$simult\_add16*, *vec\_\$simult\_add\_i*, *vec\_\$mult\_add*.

## NAME

`vec_$imult_add16` – scale and add one 16-bit integer vector to another

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$imult_add16(
    short *add_vec,
    short *mult_vec,
    long int &length,
    short &constant,
    short *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$imult_add16(
    in add_vec: univ vec_$integer16_vector;
    in mult_vec: univ vec_$integer16_vector;
    in length: integer32;
    in constant: integer16;
    out result_vec: univ vec_$integer16_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 add_vec(nvec), mult_vec(nvec), result_vec(nvec), constant
integer*4 length
```

```
call vec_$imult_add16(add_vec, mult_vec, length,
& constant, result_vec)
```

## DESCRIPTION

`Vec_$imult_add16` multiplies the vector *mult\_vec* by the scalar *constant*, adds the product to the vector *add\_vec*, and supplies the resulting vector in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = add_vec[i]
    + constant * mult_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
    result_vec[i] := add_vec[i]
                    + constant * mult_vec[i];

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
    result_vec(i) = add_vec(i)
    &              + constant * mult_vec(i)
10 continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*length* The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

When *vec\_\$imult\_add16* is used to operate on matrixes in C and Pascal, *add\_vec*, *mult\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_add*, *vec\_\$imult\_add*, *vec\_\$imult\_add16\_i*, *vec\_\$mult\_add*.

**NAME**

`vec_$simult_add16_i` – scale and add 16-bit vectors in matrixes

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$simult_add16_i(
    short *add_vec,
    long int &inc1,
    short *mult_vec,
    long int &inc2,
    long int &length,
    short &constant,
    short *result_vec,
    long int &inc3)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$simult_add16_i(
    in add_vec: univ vec_$integer16_vector;
    in inc1: integer32;
    in mult_vec: univ vec_$integer16_vector;
    in inc2: integer32;
    in length: integer32;
    in constant: integer16;
    out result_vec: univ vec_$integer16_vector;
    in inc3: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 add_vec(nvec), mult_vec(nvec), result_vec(nvec), constant
integer*4 length, inc1, inc2, inc3
```

```
call vec_$simult_add16_i(add_vec, inc1, mult_vec, inc2, length,
& constant, result_vec, inc3)
```

**DESCRIPTION**

`Vec_$simult_add16_i` multiplies *length* elements of *mult\_vec* selected by *inc2* by the scalar *constant*, adds the resulting products to elements of *add\_vec* selected by *inc1*, and supplies the results in elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use `vec_$simult_add16_i` to scale and sum individual vectors in two matrixes and place the result in a vector of another matrix. To scale and add the *N*th vector in matrix *Y* to the *M*th vector in matrix *X*, choose *inc2* equal to the number of vectors in matrix *Y* and *inc1* equal to the number of vectors in matrix *X*. Then place the *M*th element of matrix *X* at the beginning of *add\_vec*, and place the *N*th element of matrix *Y* at the beginning of *mult\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc3* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length; ++i) {
    result_vec[l] = add_vec[j]
                  + constant * mult_vec[k];
    j += inc1;
    k += inc2;
    l += inc3;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[l] := add_vec[j]
                  + constant * mult_vec[k];
    j := j + inc1;
    k := k + inc2;
    l := l + inc3;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      l = 1
      do 10 i = 1, length
         result(l) = add_vec(j)
&          + constant * mult_vec(k)
         j = j + inc1
         k = k + inc2
         l = l + inc3
10    continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*inc1*

An increment for the index of *add\_vec* that selects the elements to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*inc2*

An increment for the index of *mult\_vec* that selects the elements to multiply by *constant* and add to *add\_vec*.

*length*

The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

*inc3*

An increment for the index of *result\_vec* that selects the elements to receive the results of multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

In C and Pascal, *vec\_\$simult\_add16\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

#### SEE ALSO

*vec\_\$dmult\_add\_i*, *vec\_\$simult\_add16*, *vec\_\$simult\_add\_i*, *vec\_\$mult\_add\_i*.

## NAME

vec\_\$simult\_add\_i – scale and add 32-bit vectors in matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$simult_add_i(
    long *add_vec,
    long int &inc1,
    long *mult_vec,
    long int &inc2,
    long int &length,
    long &constant,
    long *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$simult_add_i(
    in add_vec: univ vec_$integer32_vector;
    in inc1: integer32;
    in mult_vec: univ vec_$integer32_vector;
    in inc2: integer32;
    in length: integer32;
    in constant: integer32;
    out result_vec: univ vec_$integer32_vector;
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

```
integer*4 add_vec(nvec), mult_vec(nvec), result_vec(nvec), constant
integer*4 length, inc1, inc2, inc3
```

```
call vec_$simult_add_i(add_vec, inc1, mult_vec, inc2, length,
& constant, result_vec, inc3)
```

## DESCRIPTION

Vec\_\$simult\_add\_i multiplies *length* elements of *mult\_vec* selected by *inc2* by the scalar *constant*, adds the resulting products to elements of *add\_vec* selected by *inc1*, and supplies the results in elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use `vec_simult_add_i` to scale and sum individual vectors in two matrixes and place the result in a vector of another matrix. To scale and add the *N*th vector in matrix *Y* to the *M*th vector in matrix *X*, choose *inc2* equal to the number of vectors in matrix *Y* and *inc1* equal to the number of vectors in matrix *X*. Then place the *M*th element of matrix *X* at the beginning of *add\_vec*, and place the *N*th element of matrix *Y* at the beginning of *mult\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc3* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length; ++i) {
    result_vec[l] = add_vec[j]
                  + constant * mult_vec[k];
    j += inc1;
    k += inc2;
    l += inc3;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[l] := add_vec[j]
                  + constant * mult_vec[k];
    j := j + inc1;
    k := k + inc2;
    l := l + inc3;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      l = 1
      do 10 i = 1, length
         result(l) = add_vec(j)
&          + constant * mult_vec(k)
         j = j + inc1
         k = k + inc2
         l = l + inc3
10    continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*inc1* An increment for the index of *add\_vec* that selects the elements to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*inc2* An increment for the index of *mult\_vec* that selects the elements to multiply by *constant* and add to *add\_vec*.

*length* The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

*inc3* An increment for the index of *result\_vec* that selects the elements to receive the results of multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

In C and Pascal, *vec\_\$simult\_add\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

#### SEE ALSO

*vec\_\$dmult\_add\_i*, *vec\_\$simult\_add*, *vec\_\$simult\_add16\_i*, *vec\_\$mult\_add\_i*.

**NAME**

`vec_${simult}_constant` – multiply a 32-bit integer vector by a scalar

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_${simult}_constant(
    long *mult_vec,
    long int &length,
    long &constant,
    long *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_${simult}_constant(
    in start_vec: univ vec_${integer32}_vector;
    in length: integer32;
    in constant: integer32;
    out result_vec: univ vec_${integer32}_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 mult_vec(nvec), result_vec(nvec), constant
integer*4 length
```

```
call vec_${simult}_constant(mult_vec, length, constant, result_vec)
```

**DESCRIPTION**

`Vec_${simult}_constant` multiplies the 32-bit integer array `mult_vec` by the scalar value `constant` and supplies the result in the array `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant * start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant * start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant * mult_vec(i)
10 continue

```

*mult\_vec*

The vector to multiply by *constant*.

*length* The number of elements to multiply.

*constant*

The scalar constant to multiply *mult\_vec* by.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant*.

#### NOTES

When *vec\_\$simult\_constant* is used to operate on matrixes in C and Pascal, *mult\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_constant*,  
*vec\_\$mult\_constant*.

*vec\_\$simult\_constant16*,

*vec\_\$simult\_constant\_i*,

## NAME

`vec_$simult_constant16` – multiply a 16-bit integer vector by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_$simult_constant16(
    short *mult_vec,
    long int &length,
    short &constant,
    short *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_$simult_constant16(
    in mult_vec: univ vec_$integer16_vector;
    in length: integer32;
    in constant: integer16;
    out result_vec: univ vec_$integer16_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

integer*2 mult_vec(nvec), result_vec(nvec), constant
integer*4 length

call vec_$simult_constant16(mult_vec, length, constant, result_vec)
```

## DESCRIPTION

`Vec_$simult_constant16` multiplies the 16-bit integer array `mult_vec` by the scalar value `constant` and supplies the result in the array `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant * start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant * start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant * mult_vec(i)
10 continue

```

*mult\_vec*

The vector to multiply by *constant*.

*length* The number of elements to multiply.

*constant*

The scalar constant to multiply *mult\_vec* by.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant*.

#### NOTES

When *vec\_\$simult\_constant16* is used to operate on matrixes in C and Pascal, *mult\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$mult\_constant*,  
*vec\_\$mult\_constant*.

*vec\_\$simult\_constant*,

*vec\_\$simult\_constant16\_i*,

## NAME

`vec_simult_constant16_i` – multiply a vector in a 16-bit integer matrix by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>

void vec_simult_constant16_i(
    short *mult_vec,
    long int &inc1,
    long int &length,
    short &constant,
    short *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';

procedure vec_simult_constant16_i(
    in mult_vec: univ vec_integer16_vector;
    in inc1: integer32;
    in length: integer32;
    in constant: integer16;
    out result_vec: univ vec_integer16_vector;
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'

parameter (nvec = 10)

integer*2 mult_vec(nvec), result_vec(nvec), constant
integer*4 length, inc1, inc2

call vec_simult_constant16_i(mult_vec, inc1, length, constant,
& result_vec, inc2)
```

## DESCRIPTION

`Vec_simult_constant16_i` multiplies *length* elements of the 16-bit integer array *mult\_vec* selected by *inc1* by the scalar value *constant*, and supplies the result in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_simult_constant16_i` to operate on a vector in a matrix. To multiply *M*th vector in a matrix by *constant*, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *mult\_vec*. To place the result of

the operation in the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant * mult_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := constant * mult_vec[j];
    k := k + inc2;
    j := j + incl;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
         result(j) = constant * mult_vec(k)
         j = j + inc2
         k = k + incl
10    continue

```

*mult\_vec*

The vector to multiply by *constant*.

*incl*

An increment for the index of the array *mult\_vec* that selects elements to multiply by *constant*.

*length*

The number of products to calculate.

*constant*

The scalar constant to multiply elements of *mult\_vec* by.

*result\_vec*

An array whose elements receive the product of *constant* and *mult\_vec*.

*inc2* An increment for the index of the array *result\_vec* that selects elements to receive the product of *constant* and *mult\_vec*.

**NOTES**

In C and Pascal, *vec\_\$imult\_constant16\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$dmult\_constant\_i*,            *vec\_\$imult\_constant16*,            *vec\_\$imult\_constant\_i*,  
*vec\_\$mult\_constant\_i*.

## NAME

`vec_$simult_constant_i` – multiply a vector in a 32-bit integer matrix by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$simult_constant_i(
    long *mult_vec,
    long int &inc1,
    long int &length,
    long &constant,
    long *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
#include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$simult_constant_i(
    in mult_vec: univ vec_$integer32_vector;
    in inc1: integer32;
    in length: integer32;
    in constant: integer32;
    out result_vec: univ vec_$integer32_vector;
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
#include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*4 *mult\_vec*(*nvec*), *result\_vec*(*nvec*), *constant*  
integer\*4 *length*, *inc1*, *inc2*

call `vec_$simult_constant_i`(*mult\_vec*, *inc1*, *length*, *constant*,  
& *result\_vec*, *inc2*)

## DESCRIPTION

`Vec_$simult_constant_i` multiplies *length* elements of the 32-bit integer array *mult\_vec* selected by *inc1* by the scalar value *constant* and supplies the result in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$simult_constant_i` to operate on a vector in a matrix. To multiply *M*th vector in a matrix by *constant*, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *mult\_vec*. To place the result of the operation in the

*N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant * mult_vec[j];
    k += inc2;
    j += inc1;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := constant * mult_vec[j];
    k := k + inc2;
    j := j + inc1;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result(j) = constant * mult_vec(k)
          j = j + inc2
          k = k + inc1
10    continue

```

*mult\_vec*

The vector to multiply by *constant*.

*inc1*

An increment for the index of the array *mult\_vec* that selects elements to multiply by *constant*.

*length*

The number of products to calculate.

*constant*

The scalar constant to multiply elements of *mult\_vec* by.

*result\_vec*

An array whose elements receive the product of *constant* and *mult\_vec*.

VEC\_\$SIMULT\_CONSTANT\_I

VEC\_\$SIMULT\_CONSTANT\_I

*inc2* An increment for the index of the array *result\_vec* that selects elements to receive the product of *constant* and *mult\_vec*.

**NOTES**

In C and Pascal, *vec\_\$imult\_constant\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$dmult\_constant\_i*, *vec\_\$imult\_constant16\_i*, *vec\_\$imult\_constant\_i*,  
*vec\_\$mult\_constant\_i*.

**NAME**

`vec_$init` – initialize a single-precision vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$init(
    float *vector,
    long int &length,
    float &constant)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$init(
    var vector: univ vec_$real_vector;
    in length: integer32;
    in constant: real);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *vector*(*nvec*), *constant*  
integer\*4 *length*

call `vec_$init`(*vector*, *length*, *constant*)

**DESCRIPTION**

`Vec_$init` sets *length* elements of *vector* to the single-precision value *constant*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = constant;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
    vector[i] := constant;
end
```

In FORTRAN, the resulting operation is

```
      do 10 i = 1, length
         vector(i) = constant
10    continue
```

*vector* The vector to initialize.

*length* The number of elements in *vector* to initialize.

*constant*

The value that the elements of *vector* should be set to.

#### NOTES

In C and Pascal, `vec_$init` initializes a row vector; whereas in FORTRAN, it initializes a column vector.

#### SEE ALSO

`vec_$dinit`, `vec_$iinit`, `vec_$iinit16`, `vec_$init_i`.

## NAME

`vec_$ipostmult` – multiply a 32-bit integer vector by a 4×4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$ipostmult(
    long *matrix,
    long *start_vec,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$ipostmult(
    in matrix: univ vec_$integer32_matrix;
    in start_vec: univ vec_$integer32_vector;
    out result_vec: univ vec_$integer32_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 matrix(4, 4), start_vec(4), result_vec(4)
```

```
call vec_$ipostmult(matrix, start_vec, result_vec)
```

## DESCRIPTION

`Vec_$ipostmult` multiplies the 4-element vector `start_vec` by the 4×4 matrix `matrix`.

In C, `vec_$ipostmult` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```
for (j = 0; j < 4; ++j) {
    result_vec[j] = 0;
    for (i = 0; i < 4; ++i)
        result_vec[j] += start_vec[i]
            * matrix[i][j];
}
```

In Pascal, `vec_$ipostmult` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```

for j := 1 to 4 do
  begin
    result_vec[j] := 0;
    for i := 1 to 4 do
      result_vec[j] := result_vec[j]
        + start_vec[i]
        * matrix[i, j];
    end
  end
end

```

In FORTRAN, `vec_$$postmult` applies *matrix* as a left transform to a column vector *start\_vec*, and the resulting operation is

```

do 10 j = 1, 4
  result_vec(j) = 0
  do 10 i = 1, 4
    result_vec(j) = result_vec(j)
      & + start_vec(i)
      & * matrix(j, i)
  10 continue

```

*matrix* The matrix to multiply by *start\_vec*.

*start\_vec*  
The vector to multiply by *matrix*.

*result\_vec*  
The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$$ipremult` transforms 32-bit integer vectors from the other side.

#### SEE ALSO

`vec_$$dpostmult`, `vec_$$ipostmult16`, `vec_$$postmult`.

## NAME

`vec_$(postmult16` – multiply a 16-bit integer vector by a 4x4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(postmult16(
    short *matrix,
    short *start_vec,
    short *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(postmult16(
    in matrix: univ vec_$(integer16_matrix;
    in start_vec: univ vec_$(integer16_vector;
    out result_vec: univ vec_$(integer16_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*2 matrix(4, 4), start_vec(4), result_vec(4)
```

```
call vec_$(postmult16(matrix, start_vec, result_vec)
```

## DESCRIPTION

`Vec_$(postmult16` multiplies the 4-element vector `start_vec` by the 4x4 matrix `matrix`.

In C, `vec_$(postmult16` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```
for (j = 0; j < 4; ++j) {
    result_vec[j] = 0;
    for (i = 0; i < 4; ++i)
        result_vec[j] += start_vec[i]
                        * matrix[i][j];
}
```

In Pascal, `vec_$(postmult16` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```

for j := 1 to 4 do
  begin
    result_vec[j] := 0;
    for i := 1 to 4 do
      result_vec[j] := result_vec[j]
        + start_vec[i]
        * matrix[i,j];
    end
  end
end

```

In FORTRAN, `vec_$ipostmult16` applies *matrix* as a left transform to a column vector *start\_vec*, and the resulting operation is

```

do 10 j = 1, 4
  result_vec(j) = 0
  do 10 i = 1, 4
    result_vec(j) = result_vec(j)
      & + start_vec(i)
      & * matrix(j,i)
  10 continue

```

*matrix* The matrix to multiply by *start\_vec*.

*start\_vec*  
The vector to multiply by *matrix*.

*result\_vec*  
The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$ipremult16` transforms 16-bit integer vectors from the other side.

#### SEE ALSO

`vec_$dpostmult`, `vec_$ipostmult`, `vec_$postmult`.

## NAME

`vec_$$postmultn` – multiply a 32-bit integer vector by a matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$postmultn(
    long *matrix,
    long *start_vec,
    long int &m,
    long int &n,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$postmultn(
    in matrix: univ vec_$$integer32_matrix;
    in start_vec: univ vec_$$integer32_vector;
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$$integer32_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
integer*4 matrix(m, n), start_vec(n), result_vec(m)
```

```
call vec_$$postmultn(matrix, start_vec, m, n, result_vec)
```

## DESCRIPTION

`Vec_$$postmultn` multiplies the  $n$ -element vector `start_vec` by the variably dimensioned matrix `matrix`, and supplies the resulting  $m$ -element vector in `result_vec`.

In C, `vec_$$postmultn` applies the  $n \times m$  matrix `matrix` as a right transform to the  $m$ -element row vector `start_vec`, and supplies the transformed  $n$ -element result in `result_vec`:

```

for (i = 0; i < m; ++i) {
    result_vec[i] = 0;
    for (j = 0; j < n; ++j)
        result_vec[i] += start_vec[j]
                        * matrix[j][i];
}

```

In Pascal, `vec_$ipostmultn` applies the  $n \times m$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to m do
begin
    result_vec[i] := 0.0;
    for j := 1 to n do
        result_vec[i] := result_vec[i]
                        + start_vec[j]
                        * matrix[j,i];
    end;
end;

```

In FORTRAN, `vec_$ipostmultn` applies the  $m \times n$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, m
    result_vec(i) = 0.0
    do 10 j = 1, n
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(i,j)
    &
    &
10 continue

```

*matrix* A matrix to multiply *start\_vec* by.

*start\_vec* An  $n$ -element vector to multiply by *matrix*.

*m* The number of elements in *start\_vec*.

*n* The number of elements in *result\_vec*.

*result\_vec* An  $m$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$ipremultn` transforms 32-bit integer vectors from the other side.

#### SEE ALSO

`vec_$dpostmultn`, `vec_$ipostmultn16`, `vec_$postmultn`.

**NAME**

**vec\_\$(postmultn16** – multiply a 16-bit integer vector by a matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(postmultn16(
    short *matrix,
    short *start_vec,
    long int &m,
    long int &n,
    short *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(postmultn16(
    in matrix: univ vec_$(integer16_matrix);
    in start_vec: univ vec_$(integer16_vector);
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$(integer16_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
integer*2 matrix(m, n), start_vec(n), result_vec(m)
```

```
call vec_$(postmultn16(matrix, start_vec, m, n, result_vec)
```

**DESCRIPTION**

**Vec\_\$(postmultn16** multiplies the  $n$ -element vector *start\_vec* by the variably dimensioned matrix *matrix*, and supplies the resulting  $m$ -element vector in *result\_vec*.

In C, **vec\_\$(postmultn16** applies the  $n \times m$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for (i = 0; i < m; ++i) {
    result_vec[i] = 0;
    for (j = 0; j < n; ++j)
        result_vec[i] += start_vec[j]
            * matrix[j][i];
}

```

In Pascal, `vec_$(postmultn16)` applies the  $n \times m$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to m do
begin
    result_vec[i] := 0.0;
    for j := 1 to n do
        result_vec[i] := result_vec[i]
            + start_vec[j]
            * matrix[j,i];
end;

```

In FORTRAN, `vec_$(postmultn16)` applies the  $m \times n$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, m
    result_vec(i) = 0.0
    do 10 j = 1, n
        result_vec(i) = result_vec(i)
            &                + start_vec(j)
            &                * matrix(i,j)
10 continue

```

*matrix* A matrix to multiply *start\_vec* by.

*start\_vec* An  $n$ -element vector to multiply by *matrix*.

*m* The number of elements in *start\_vec*.

*n* The number of elements in *result\_vec*.

*result\_vec* An  $m$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$(ipremultn16)` transforms 16-bit integer vectors from the other side.

#### SEE ALSO

`vec_$(dpostmultn)`, `vec_$(postmultn)`, `vec_$(postmultn)`.

## NAME

`vec_$ipremult` – multiply a 32-bit vector by a 4×4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$ipremult(
    long *start_vec,
    long *matrix,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$ipremult(
    in start_vec: univ vec_$integer32_vector;
    in matrix: univ vec_$integer32_matrix;
    out result_vec: univ vec_$integer32_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 start_vec(4), matrix(4,4), result_vec(4)
```

```
call vec_$ipremult(start_vec, matrix, result_vec)
```

## DESCRIPTION

`Vec_$ipremult` multiplies the 4-element vector `start_vec` by the 4×4 matrix `matrix`.

In C, `vec_$ipremult` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```
for (i = 0; i < 4; ++i) {
    result_vec[i] = 0;
    for (j = 0; j < 4; ++j)
        result_vec[i] += start_vec[j]
            * matrix[i][j];
}
```

In Pascal, `vec_$ipremult` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```

for i := 1 to 4 do
  begin
    result_vec[i] := 0;
    for j := 1 to 4 do
      result_vec[i] := result_vec[i]
        + start_vec[i]
        * matrix[i,j];
    end
  end

```

In FORTRAN, `vec_$ipremult` applies *matrix* as a right transform to a row vector *start\_vec*, and the resulting operation is

```

do 10 i = 1, 4
  result_vec(i) = 0
  do 10 j = 1, 4
    result_vec(i) = result_vec(i)
      & + start_vec(j)
      & * matrix(j,i)
  10 continue

```

*start\_vec*

The vector to multiply by *matrix*.

*matrix* The matrix to multiply by *start\_vec*.

*result\_vec*

The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$ipostmult` transforms 32-bit integer vectors from the other side.

#### SEE ALSO

`vec_$dpremult`, `vec_$ipremult16`, `vec_$premult`.

## NAME

`vec_$(premult16` – multiply a 16-bit vector by a 4×4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(premult16(
    short *start_vec,
    short *matrix,
    short *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(premult16(
    in start_vec: univ vec_$(integer16_vector);
    in matrix: univ vec_$(integer16_matrix);
    out result_vec: univ vec_$(integer16_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*2 start_vec(4), matrix(4,4), result_vec(4)
```

```
call vec_$(premult16(start_vec, matrix, result_vec)
```

## DESCRIPTION

`Vec_$(premult16` multiplies the 4-element vector `start_vec` by the 4×4 matrix `matrix`.

In C, `vec_$(premult16` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```
for (i = 0; i < 4; ++i) {
    result_vec[i] = 0;
    for (j = 0; j < 4; ++j)
        result_vec[i] += start_vec[j]
            * matrix[i][j];
}
```

In Pascal, `vec_$(premult16` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```

for i := 1 to 4 do
  begin
    result_vec[i] := 0;
    for j := 1 to 4 do
      result_vec[i] := result_vec[i]
                      + start_vec[i]
                      * matrix[i,j];
    end
  end

```

In FORTRAN, `vec_$ipremult16` applies *matrix* as a right transform to a row vector *start\_vec*, and the resulting operation is

```

do 10 i = 1, 4
  result_vec(i) = 0
  do 10 j = 1, 4
    result_vec(i) = result_vec(i)
                  + start_vec(j)
                  * matrix(j,i)
  &
  &
10 continue

```

*start\_vec*

The vector to multiply by *matrix*.

*matrix* The matrix to multiply by *start\_vec*.

*result\_vec*

The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$ipostmult16` transforms 16-bit integer vectors from the other side.

#### SEE ALSO

`vec_$dpremult`, `vec_$ipremult`, `vec_$premult`.

## NAME

`vec_$(premultn` – multiply a 32-bit integer vector by a matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(premultn(
    long *start_vec,
    long *matrix,
    long int &m,
    long int &n,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(premultn(
    in start_vec: univ vec_$(integer32_vector);
    in matrix: univ vec_$(integer32_matrix);
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$(integer32_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
integer*4 start_vec(m), matrix(m, n), result_vec(n)
```

```
call vec_$(premultn(start_vec, matrix, m, n, result_vec)
```

## DESCRIPTION

`Vec_$(premultn` multiplies the  $m$ -element vector `start_vec` by the variably dimensioned matrix `matrix`, and supplies the resulting  $n$ -element vector in `result_vec`.

In C, `vec_$(premultn` applies the  $n \times m$  matrix `matrix` as a left transform to the  $m$ -element column vector `start_vec`, and supplies the transformed  $n$ -element result in `result_vec`:

```

for (i = 0; i < n; ++i) {
    result_vec[i] = 0;
    for (j = 0; j < m; ++j)
        result_vec[i] += start_vec[j]
            * matrix[i][j];
}

```

In Pascal, `vec_$$premultn` applies the  $n \times m$  matrix *matrix* as a left transform to the *m*-element column vector *start\_vec*, and supplies the transformed *n*-element result in *result\_vec*:

```

for i := 1 to n do
begin
    result_vec[i] := 0;
    for j := 1 to m do
        result_vec[i] := result_vec[i]
            + start_vec[j]
            * matrix[i, j];
    end
end

```

In FORTRAN, `vec_$$premultn` applies the  $m \times n$  matrix *matrix* as a right transform to the *m*-element row vector *start\_vec*, and supplies the transformed *n*-element result in *result\_vec*:

```

do 10 i = 1, n
    result_vec(i) = 0
    do 10 j = 1, m
        result_vec(i) = result_vec(i)
&                + start_vec(j)
&                * matrix(j, i)
10 continue

```

*start\_vec*

An *m*-element vector to multiply by *matrix*.

*matrix* A matrix to multiply *start\_vec* by.

*m* The number of elements in *start\_vec*.

*n* The number of elements in *result\_vec*.

*result\_vec*

An *n*-element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$$postmultn` transforms 32-bit integer vectors from the other side.

#### SEE ALSO

`vec_$$dpremultn`, `vec_$$premultn16`, `vec_$$premultn`.

**NAME**

**vec\_\$(premultn16** – multiply a 16-bit integer vector by a matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$(premultn16(
    short *start_vec,
    short *matrix,
    long int &m,
    long int &n,
    short *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$(premultn16(
    in start_vec: univ vec_$(integer16_vector);
    in matrix: univ vec_$(integer16_matrix);
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$(integer16_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
integer*2 start_vec(m), matrix(m, n), result_vec(n)
```

```
call vec_$(premultn16(start_vec, matrix, m, n, result_vec)
```

**DESCRIPTION**

**Vec\_\$(premultn16** multiplies the  $m$ -element vector *start\_vec* by the variably dimensioned matrix *matrix*, and supplies the resulting  $n$ -element vector in *result\_vec*.

In C, **vec\_\$(premultn16** applies the  $n \times m$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for (i = 0; i < n; ++i) {
    result_vec[i] = 0;
    for (j = 0; j < m; ++j)
        result_vec[i] += start_vec[j]
                        * matrix[i][j];
}

```

In Pascal, `vec_$$premultn16` applies the  $n \times m$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to n do
begin
    result_vec[i] := 0;
    for j := 1 to m do
        result_vec[i] := result_vec[i]
                        + start_vec[j]
                        * matrix[i,j];
    end
end

```

In FORTRAN, `vec_$$premultn16` applies the  $m \times n$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, n
    result_vec(i) = 0
    do 10 j = 1, m
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(j,i)
    &
    &
10 continue
start_vec

```

An  $m$ -element vector to multiply by *matrix*.

*matrix* A matrix to multiply *start\_vec* by.

*m* The number of elements in *start\_vec*.

*n* The number of elements in *result\_vec*.

*result\_vec*

An  $n$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$$postmultn16` transforms 16-bit integer vectors from the other side.

#### SEE ALSO

`vec_$$dpremultn`, `vec_$$premultn`, `vec_$$premultn`.

## NAME

`vec_$isub` – subtract 32-bit integer vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$isub(
    long *start_vec,
    long *sub_vec,
    long int &length,
    long *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$isub(
    in start_vec: univ vec_$integer32_vector;
    in sub_vec: univ vec_$integer32_vector;
    in length: integer32;
    out result_vec: univ vec_$integer32_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*4 start_vec(nvec), sub_vec(nvec), result_vec(nvec)
integer*4 length
```

```
call vec_$isub(start_vec, sub_vec, length, result_vec)
```

## DESCRIPTION

`Vec_$isub` subtracts *length* elements of *sub\_vec* from *start\_vec* and supplies the difference in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] - sub_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    result_vec[i] := start_vec[i] - sub_vec[i];
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i) - sub_vec(i)
10 continue
```

*start\_vec*

The vector to subtract *sub\_vec* from.

*sub\_vec* The vector to subtract from *start\_vec*.

*length* The number of differences to calculate.

*result\_vec*

The difference of *start\_vec* and *sub\_vec*.

#### NOTES

When *vec\_\$isub* is used to operate on matrixes in C and Pascal, *start\_vec*, *result\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dsub*, *vec\_\$isub16*, *vec\_\$isub\_i*, *vec\_\$sub*.

**NAME**

`vec_${isub16}` – subtract 16-bit integer vectors

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_${isub16}(
    short *start_vec,
    short *sub_vec,
    long int &length,
    short *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_${isub16}(
    in start_vec: univ vec_${integer16}_vector;
    in sub_vec: univ vec_${integer16}_vector;
    in length: integer32;
    out result_vec: univ vec_${integer16}_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 start_vec(nvec), sub_vec(nvec), result_vec(nvec)
integer*4 length
```

call `vec_${isub16}(start_vec, sub_vec, length, result_vec)`

**DESCRIPTION**

`Vec_${isub16}` subtracts *length* elements of *sub\_vec* from *start\_vec* and supplies the difference in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] - sub_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    result_vec[i] := start_vec[i] - sub_vec[i];
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i) - sub_vec(i)
10 continue
```

*start\_vec*

The vector to subtract *sub\_vec* from.

*sub\_vec* The vector to subtract from *start\_vec*.

*length* The number of differences to calculate.

*result\_vec*

The difference of *start\_vec* and *sub\_vec*.

#### NOTES

When *vec\_\$isub16* is used to operate on matrixes in C and Pascal, *start\_vec*, *result\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dsub*, *vec\_\$isub16\_i*, *vec\_\$isub\_i*, *vec\_\$sub*.

## NAME

vec\_\${isub16}\_i – subtract 16-bit integer vectors in matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 start_vec(nvec), sub_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3
```

```
call vec_${isub16}_i(start_vec, inc1, sub_vec, inc2,
& length, result_vec, inc3)
```

## DESCRIPTION

Vec\_\${isub16}\_i subtracts *length* elements of *sub\_vec* selected by *inc2* from *length* elements of *start\_vec* selected by *inc1*, and supplies the result in the elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use vec\_\${isub16}\_i to subtract individual vectors in two matrixes and place the difference in a vector of another matrix. To subtract the *N*th vector in matrix *Y* from the *M*th vector in matrix *X*, choose *inc2* equal to the number of vectors in matrix *Y* and *inc1* equal to the number of vectors in matrix *X*. Then place the *M*th element of matrix *X* at the beginning of *start\_vec*, and place the *N*th element of matrix *Y* at the beginning of *sub\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc3* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length, ++i) {
    result_vec[j] = start_vec[k] - sub_vec[l];
    j += inc3;
    k += inc1;
    l += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] - sub_vec[l];
    j := j + inc3;
    k := k + inc1;
    l := l + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
l = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) - sub_vec(l)
    j = j + inc3
    k = k + inc1
    l = l + inc2
10 continue

```

*start\_vec*

The vector to subtract *sub\_vec* from.

*inc1*

An increment for the index of *start\_vec* that chooses which elements the elements of *sub\_vec* will be subtracted from.

*sub\_vec*

The vector to subtract from *start\_vec*.

- inc2* An increment for the index of *sub\_vec* that chooses which elements will be subtracted from the elements of *start\_vec*.
- length* The number of scalar differences to calculate.
- result\_vec*  
The difference of *start\_vec* and *sub\_vec*.
- inc3* An increment for the index of *result\_vec* that chooses which elements will received the differences.

**NOTES**

In C and Pascal, *vec\_\$isub16\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$dsub\_i*, *vec\_\$isub16*, *vec\_\$isub\_i*, *vec\_\$sub\_i*.

**NAME**

vec\_\${isub}\_i – subtract 32-bit integer vectors in matrixes

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_${isub}_i(
    long *start_vec,
    long int &inc1,
    long *sub_vec,
    long int &inc2,
    long int &length,
    long *result_vec,
    long int &inc3)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
#include '/sys/ins/vec.ins.pas';
```

```
procedure vec_${isub}_i(
    in start_vec: univ vec_${integer32}_vector;
    in inc1: integer32;
    in sub_vec: univ vec_${integer32}_vector;
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_${integer32}_vector;
    in inc3: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
#include '/sys/ins/vec.ins.ftn'
```

parameter (nvec = 10)

```
integer*4 start_vec(nvec), sub_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3
```

```
call vec_${isub}_i(start_vec, inc1, sub_vec, inc2,
& length, result_vec, inc3)
```

**DESCRIPTION**

Vec\_\${isub}\_i subtracts *length* elements of *sub\_vec* selected by *inc2* from *length* elements of *start\_vec* selected by *inc1*, and supplies the result in the elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use vec\_\${isub}\_i to subtract individual vectors in two matrixes and place the difference in a vector of another

matrix. To subtract the  $N$ th vector in matrix  $Y$  from the  $M$ th vector in matrix  $X$ , choose  $inc2$  equal to the number of vectors in matrix  $Y$  and  $inc1$  equal to the number of vectors in matrix  $X$ . Then place the  $M$ th element of matrix  $X$  at the beginning of  $start\_vec$ , and place the  $N$ th element of matrix  $Y$  at the beginning of  $sub\_vec$ . To place the result of the operation in the  $P$ th vector of a resultant matrix, choose  $inc3$  equal to the number of vectors in the resultant matrix, and place the  $P$ th element of the matrix array at the beginning of  $result\_vec$ .

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length, ++i) {
    result_vec[j] = start_vec[k] - sub_vec[l];
    j += inc3;
    k += inc1;
    l += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] - sub_vec[l];
    j := j + inc3;
    k := k + inc1;
    l := l + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
l = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) - sub_vec(l)
    j = j + inc3
    k = k + inc1
    l = l + inc2
10 continue

```

- start\_vec* The vector to subtract *sub\_vec* from.
- inc1* An increment for the index of *start\_vec* that chooses which elements the elements of *sub\_vec* will be subtracted from.
- sub\_vec* The vector to subtract from *start\_vec*.
- inc2* An increment for the index of *sub\_vec* that chooses which elements will be subtracted from the elements of *start\_vec*.
- length* The number of scalar differences to calculate.
- result\_vec*  
The difference of *start\_vec* and *sub\_vec*.
- inc3* An increment for the index of *result\_vec* that chooses which elements will received the differences.

**NOTES**

In C and Pascal, *vec\_\$isub\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$dsb\_i*, *vec\_\$isub*, *vec\_\$isub16\_i*, *vec\_\$sub\_i*.

**NAME**

`vec_$isum` – sum the elements of a 32-bit integer vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
long vec_$isum(
    long *vec,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$isum(
    in vec: univ vec_$integer32_vector;
    in length: integer32): integer32
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

**parameter** (*nvec* = 10)

**integer\*4** *vec(nvec)*, *sum*  
**integer\*4** *length*

*sum* = `vec_$isum`(*vec*, *length*)

**DESCRIPTION**

`Vec_$isum` adds together *length* elements of the 32-bit integer array *vector* and returns the sum.

In C, the resulting operation is

```
return_value = 0;
for (i = 0; i < length; ++i)
    return_value += vec[i];
```

In Pascal, the resulting operation is

```
return_value := 0;
for i := 1 to length do
    return_value := return_value + vec[i];
```

In FORTRAN, the resulting operation is

```
      vec_$isum = 0
      do 10 i = 1, length
          vec_$isum = vec_$isum + vec(i)
10    continue
```

*vec* The vector to sum.

*length* The number of elements in *vec* to sum.

#### NOTES

When *vec\_\$isum* is used to operate on matrixes in C and Pascal, *vec* is a row vector; whereas in FORTRAN, it is a column vector.

#### SEE ALSO

*vec\_\$dsum*, *vec\_\$isum16*, *vec\_\$isum\_i*, *vec\_\$sum*.

## NAME

`vec_$$sum16` – sum the elements of a 16-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
short vec_$$sum16(
    short *vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$$sum16(
    in vec: univ vec_$$integer16_vector;
    in length: integer32): integer16;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 vec(nvec), sum
integer*4 length
```

```
sum = vec_$$sum16(vec, length)
```

## DESCRIPTION

`Vec_$$sum16` adds together *length* elements of the 16-bit integer array *vector* and returns the sum.

In C, the resulting operation is

```
return_value = 0;
for (i = 0; i < length; ++i)
    return_value += vec[i];
```

In Pascal, the resulting operation is

```
return_value := 0;
for i := 1 to length do
    return_value := return_value + vec[i];
```

In FORTRAN, the resulting operation is

```
      vec_$isum16 = 0
      do 10 i = 1, length
          vec_$isum16 = vec_$isum16 + vec(i)
10    continue
```

*vec* The vector to sum.

*length* The number of elements in *vec* to sum.

#### NOTES

When *vec\_\$isum16* is used to operate on matrixes in C and Pascal, *vec* is a row vector; whereas in FORTRAN, it is a column vector.

#### SEE ALSO

*vec\_\$dsum*, *vec\_\$isum*, *vec\_\$isum16\_i*, *vec\_\$sum*.

**NAME**

`vec_${isum16}_i` – sum the elements of a vector in a 16-bit integer matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
short vec_${isum16}_i(
    short *vec,
    long int &inc,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_${isum16}_i(
    in vec: univ vec_${integer16}_vector;
    in inc: integer32;
    in length: integer32): integer16;
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*2 *vec(nvec)*, *sum*  
integer\*4 *length*, *inc*

*sum* = `vec_${isum16}_i`(*vec*, *inc*, *length*)

**DESCRIPTION**

`Vec_${isum16}_i` adds *length* elements from the 16-bit integer array *vector* selected by *inc* and returns the sum.

Through appropriate choice of *inc*, a program can use `vec_${isum16}_i` to sum an individual vector in a matrix. To sum the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix.

In C, the resulting operation is

```
return_value = 0;
j = 0;
for (i = 0; i < length; ++i) {
    return_value += vec[j];
    j += inc;
}
```

In Pascal, the resulting operation is

```
return_value := 0;
j := 1;
for i := 1 to length do
begin
    return_value := return_value + vec[j];
    j := j + inc;
end
```

In FORTRAN, the resulting operation is

```
vec_$isum16_i = 0
j = 1
do 10 i = 1, length
    vec_$isum16_i = vec_$isum16_i + vec(j)
    j = j + inc
10 continue
```

*vec* An array that contains the elements to sum.

*inc* An increment for the index of *vec* that selects which elements of *vec* are summed.

*length* The number of elements in *vec* to sum.

#### NOTES

In C and Pascal, *vec\_\$isum16\_i* sums a column vector; whereas in FORTRAN, it sums a row vector.

#### SEE ALSO

*vec\_\$dsum\_i*, *vec\_\$isum16*, *vec\_\$isum\_i*, *vec\_\$sum\_i*.

## NAME

`vec_$isum_i` – sum the elements of a vector in a 32-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
long vec_$isum_i(
    long *vec,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$isum_i(
    in vec: univ vec_$integer32_vector;
    in inc: integer32;
    in length: integer32): integer32;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*4 vec(nvec), sum
integer*4 length, inc
```

```
sum = vec_$isum_i(vec, inc, length)
```

## DESCRIPTION

`Vec_$isum_i` adds *length* elements from the 32-bit integer array *vector* selected by *inc* and returns the sum.

Through appropriate choice of *inc*, a program can use `vec_$isum_i` to sum an individual vector in a matrix. To sum the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix.

In C, the resulting operation is

```
return_value = 0;
j = 0;
for (i = 0; i < length; ++i) {
    return_value += vec[j];
    j += inc;
}
```

In Pascal, the resulting operation is

```
return_value := 0;
j := 1;
for i := 1 to length do
begin
    return_value := return_value + vec[j];
    j := j + inc;
end
```

In FORTRAN, the resulting operation is

```
vec_$isum_i = 0
j = 1
do 10 i = 1, length
    vec_$isum_i = vec_$isum_i + vec(j)
    j = j + inc
10 continue
```

*vec* An array that contains the elements to sum.

*inc* An increment for the index of *vec* that selects which elements of *vec* are summed.

*length* The number of elements in *vec* to sum.

#### NOTES

In C and Pascal, *vec\_\$isum\_i* sums a column vector; whereas in FORTRAN, it sums a row vector.

#### SEE ALSO

*vec\_\$dsum\_i*, *vec\_\$isum*, *vec\_\$isum16\_i*, *vec\_\$sum\_i*.

## NAME

`vec_$$iswap` – swap two 32-bit integer vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$$iswap(
    long *vec1,
    long *vec2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$$iswap(
    var vec1: univ vec_$$integer32_vector;
    var vec2: univ vec_$$integer32_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 vec1(nvec), vec2(nvec)
integer*4 length
```

```
call vec_$$iswap(vec1, vec2, length)
```

## DESCRIPTION

`Vec_$$iswap` swaps *length* elements between the 32-bit integer vectors *vec1* and *vec2*.

In C, the resulting operation is

```
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
}
```

In Pascal, the resulting operation is

```

for i := 1 to length do
begin
temp := vec1[i];
vec1[i] := vec2[i];
vec2[i] := temp;
end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
temp = vec1(i)
vec1(i) = vec2(i)
vec2(i) = temp
10 continue
vec1 The vector to be swapped with vec2.
vec2 The vector to be swapped with vec1.
length The number of elements to swap.

```

#### NOTES

When `vec_$iswap` is used to operate on matrixes in C and Pascal, `vec1` and `vec2` are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

`vec_$dswap`, `vec_$iswap16`, `vec_$iswap_i`, `vec_$swap`.

## NAME

`vec_$iswap16` – swap two 16-bit integer vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$iswap16(
    short *vec1,
    short *vec2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$iswap16(
    var vec1: univ vec_$integer16_vector;
    var vec2: univ vec_$integer16_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*2 vec1(nvec), vec2(nvec)
integer*4 length
```

```
call vec_$iswap16(vec1, vec2, length)
```

## DESCRIPTION

`Vec_$iswap16` swaps *length* elements between the 16-bit integer vectors *vec1* and *vec2*.

In C, the resulting operation is

```
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
}
```

In Pascal, the resulting operation is

```
for i := 1 to length do
begin
temp := vec1[i];
vec1[i] := vec2[i];
vec2[i] := temp;
end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
temp = vec1(i)
vec1(i) = vec2(i)
vec2(i) = temp
10 continue
vec1 The vector to be swapped with vec2.
vec2 The vector to be swapped with vec1.
length The number of elements to swap.
```

#### NOTES

When `vec_$iswap16` is used to operate on matrixes in C and Pascal, `vec1` and `vec2` are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

`vec_$swap`, `vec_$dswap`, `vec_$iswap`, `vec_$iswap16`.

**NAME**

`vec_${iswap16}_i` – swap two vectors in a 16-bit integer matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_${iswap16}_i(
    short *vec1,
    long int &inc1,
    short *vec2,
    long int &inc2,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_${iswap16}_i(
    var vec1: univ vec_${integer16}_vector;
    in inc1: integer32;
    var vec2: univ vec_${integer16}_vector;
    in inc2: integer32;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*2 vec1(nvec), vec2(nvec)
integer*4 length, inc1, inc2
```

```
call vec_${iswap16}_i(vec1, inc1, vec2, inc2, length)
```

**DESCRIPTION**

`vec_${iswap16}_i` swaps the *length* elements of *vec1* selected by *inc1* with the *length* elements of *vec2* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_${iswap16}_i` to swap vectors between two matrixes. To select the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vec1*. To swap the selected vector with the *N*th vector of the same or another matrix, choose *inc2* equal to the number of vectors in the same or other matrix, and place the *N*th element of the matrix array at the beginning of *vec2*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
    j += inc1;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    temp := vec1[i];
    vec1[i] := vec2[i];
    vec2[i] := temp;
    j := j + inc1;
    k := k + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
do 10 i = 1, length
    temp = vec1(j)
    vec1(j) = vec2(k)
    vec2(k) = temp
    j = j + inc1
    k = k + inc2
10 continue

```

*vec1* The vector to be swapped with *vec2*.

*inc1* The increment for the index of *vec1* that selects the elements to be swapped with *vec2*.

*vec2* The vector to be swapped with *vec1*.

*inc2* The increment for the index of *vec2* that selects the elements to be swapped with *vec1*.

*length* The number of elements to swap.

**NOTES**

In C and Pascal, `vec_$iswap16_i` swaps column vectors; whereas in FORTRAN, it swaps row vectors.

**SEE ALSO**

`vec_$dswap_i`, `vec_$iswap16`, `vec_$iswap_i`, `vec_$swap_i`.

## NAME

`vec_$iswap_i` – swap two vectors in a 32-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$iswap_i(
    long *vec1,
    long int &incl,
    long *vec2,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$iswap_i(
    var vec1: univ vec_$integer32_vector;
    in incl: integer32;
    var vec2: univ vec_$integer32_vector;
    in inc2: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

integer\*4 *vec1*(*nvec*), *vec2*(*nvec*)  
integer\*4 *length*, *incl*, *inc2*

call `vec_$iswap_i`(*vec1*, *incl*, *vec2*, *inc2*, *length*)

## DESCRIPTION

`Vec_$iswap_i` swaps the *length* elements of *vec1* selected by *incl* with the *length* elements of *vec2* selected by *inc2*.

Through appropriate choice of *incl* and *inc2*, a program can use `vec_$iswap_i` to swap vectors between two matrixes. To select the *M*th vector in a matrix, choose *incl* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vec1*. To swap the selected vector with the *N*th vector of the same or another matrix, choose *inc2* equal to the number of vectors in the same or other matrix, and place the *N*th element of the matrix array at the beginning of *vec2*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
    j += incl;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    temp := vec1[i];
    vec1[i] := vec2[i];
    vec2[i] := temp;
    j := j + incl;
    k := k + inc2;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          temp = vec1(j)
          vec1(j) = vec2(k)
          vec2(k) = temp
          j = j + incl
          k = k + inc2
10    continue

```

*vec1* The vector to be swapped with *vec2*.

*incl* The increment for the index of *vec1* that selects the elements to be swapped with *vec2*.

*vec2* The vector to be swapped with *vec1*.

*inc2* The increment for the index of *vec2* that selects the elements to be swapped with *vec1*.

VEC\_\$ISWAP\_I

VEC\_\$ISWAP\_I

*length* The number of elements to swap.

**NOTES**

In C and Pascal, `vec_$iswap_i` swaps column vectors; whereas in FORTRAN, it swaps row vectors.

**SEE ALSO**

`vec_$dswap_i`, `vec_$iswap`, `vec_$iswap16_i`, `vec_$swap_i`.

## NAME

`vec_$izero` – zero a 32-bit integer vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$izero(
    long *vector,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$izero
    var vector: univ vec_$integer32_vector;
    in length: integer32;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 vector(nvec)
integer*4 length
```

```
call vec_$izero(vector, length)
```

## DESCRIPTION

`Vec_$izero` zeros the first *length* elements of the 32-bit integer vector *vector*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = 0;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    vector[i] := 0;
```

In FORTRAN, the resulting operation is

```
      do 10 i = 1, length
         vec(i) = 0
10    continue
vector  The vector to be zeroed.
length  The number of elements in vector to zero.
```

**NOTES**

In C and Pascal, `vec_sizezero` zeros row vectors; whereas in FORTRAN, it zeros column vectors.

**SEE ALSO**

`vec_szzero`, `vec_sizezero16`, `vec_sizezero_i`, `vec_szzero`.

**NAME**

`vec_$zero16` – zero a 16-bit integer vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$zero16(
    short *vector,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$zero16(
    var vector: univ vec_$integer16_vector;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*2 vector(nvec)
integer*4 length
```

```
call vec_$zero16(vector, length)
```

**DESCRIPTION**

`Vec_$zero16` zeros the first *length* elements of the 16-bit integer vector *vector*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = 0;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    vector[i] := 0;
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    vec(i) = 0
10 continue
```

*vector* The vector to be zeroed.  
*length* The number of elements in *vector* to zero.

**NOTES**

In C and Pascal, `vec_$zero16` zeros row vectors; whereas in FORTRAN, it zeros column vectors.

**SEE ALSO**

`vec_$dzero`, `vec_$zero`, `vec_$zero16_i`, `vec_$zero`.

## NAME

vec\_\$zero16\_i – zero a vector in a 16-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$zero16_i(
    short *vector,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$zero16_i
    var vector: univ vec_$integer16_vector;
    in inc: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
integer*2 vector(nvec)
integer*4 length, inc
```

```
call vec_$zero16_i(vector, inc, length)
```

## DESCRIPTION

Vec\_\$zero16\_i zeros the *length* elements of the 16-bit integer array *vector* selected by *inc*.

Through appropriate choice of *inc*, a program can use vec\_\$zero16\_i to zero a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

j = 0;
for (i = 0; i < length; ++i) {
    vector[i] = 0;
    j += inc;
}

```

In Pascal, the resulting operation is

```

j := 1;
for i := 1 to length do
begin
    vector[i] := 0;
    j := j + inc;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      do 10 i = 1, length
          vec(j) = 0
          j = j + inc
10    continue

```

*vector* The vector to be zeroed.

*inc* An increment for the index of *vector* that chooses the elements to be zeroed.

*length* The number of elements in *vector* to zero.

#### NOTES

In C and Pascal, `vec_$zero16_i` zeros column vectors; whereas in FORTRAN, it zeros row vectors.

#### SEE ALSO

`vec_$dzero_i`, `vec_$zero16`, `vec_$zero_i`, `vec_$zero_i`.

## NAME

vec\_\$izero\_i – zero a vector in a 32-bit integer matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$izero_i(
    long *vector,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$izero_i(
    var vector: univ vec_$integer32_vector;
    in inc: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
integer*4 vector(nvec)
integer*4 length, inc
```

```
call vec_$izero_i(vector, inc, length)
```

## DESCRIPTION

Vec\_\$izero\_i zeros the *length* elements of the 32-bit integer array *vector* selected by *inc*.

Through appropriate choice of *inc*, a program can use vec\_\$izero\_i to zero a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```
j = 0;
for (i = 0; i < length; ++i) {
    vector[i] = 0;
    j += inc;
}
```

In Pascal, the resulting operation is

```

j := 1;
for i := 1 to length do
  begin
    vector[i] := 0;
    j := j + inc;
  end

```

In FORTRAN, the resulting operation is

```

      j = 1
      do 10 i = 1, length
          vec(j) = 0
          j = j + inc
10    continue

```

*vector* The vector to be zeroed.

*inc* An increment for the index of *vector* that chooses the elements to be zeroed.

*length* The number of elements in *vector* to zero.

#### NOTES

In C and Pascal, *vec\_\$zero\_i* zeros column vectors; whereas in FORTRAN, it zeros row vectors.

#### SEE ALSO

*vec\_\$dzero\_i*, *vec\_\$zero*, *vec\_\$zero16\_i*, *vec\_\$zero\_i*.

## NAME

`vec_$mat_mult` – multiply two 4×4 single-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$mat_mult(
    float *matrix1,
    float *matrix2,
    float *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$mat_mult(
    in matrix1: univ vec_$real_matrix;
    in matrix2: univ vec_$real_matrix;
    out out_matrix: univ vec_$real_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
real matrix1(4, 4), matrix2(4, 4), out_matrix(4, 4)
```

```
call vec_$mat_mult(matrix1, matrix2, out_matrix)
```

## DESCRIPTION

`Vec_$mat_mult` multiplies two 4×4 matrixes, *matrix1* and *matrix2*, and supplies the result in *out\_matrix*.

In C, `vec_$mat_mult` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```
for (i = 0; i < 4; ++i)
    for (j = 0; j < 4; ++j) {
        out_mat[j, i] = 0.0;
        for (k = 0; k < 4; ++k)
            out_mat[j][i] += matrix1[k][i]
                * matrix2[j][k];
    }
```

In Pascal, `vec_$mat_mult` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to 4 do
  for j := 1 to 4 do
    begin
      out_mat[j,i] := 0.0;
      for k := 1 to 4 do
        out_mat[j,i] := out_mat[j,i]
          + matrix1[k,i]
          * matrix2[j,k];
      end;
    end;
  end;
end;

```

In FORTRAN, `vec_$mat_mult` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, 4
  do 10 j = 1, 4
    out_mat(i,j) = 0.0
    do 10 k = 1, 4
      out_mat(i,j) = out_mat(i,j)
        + matrix1(i,k)
        * matrix2(k,j)
    &
  &
10 continue

```

*matrix1* A 4x4 matrix.  
*matrix2* Another 4x4 matrix.  
*out\_matrix*  
 The product of *matrix1* and *matrix2*.

#### NOTES

`Vec_$mat_multn` performs the same operations for variably dimensioned matrices.

#### SEE ALSO

`vec_$dmat_mult`, `vec_$imat_mult`, `vec_$imat_mult16`.

## NAME

`vec_$mat_multn` – multiply two single-precision matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$mat_multn(
    float *matrix1,
    float *matrix2,
    long int &m,
    long int &n,
    long int &s,
    float *out_matrix)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$mat_multn(
    in matrix1: univ vec_$real_matrix;
    in matrix2: univ vec_$real_matrix;
    in m: integer32;
    in n: integer32;
    in s: integer32;
    out out_matrix: univ vec_$real_matrix);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n, s
parameter (m = 3, n = 4, s = 5)
```

```
real matrix1(m, n), matrix2(n, s), out_matrix(m, s)
```

```
call vec_$mat_multn(matrix1, matrix2, m, n, s, out_matrix)
```

## DESCRIPTION

`Vec_$mat_multn` multiplies two variably dimensioned, single-precision matrixes, `matrix1` and `matrix2`, and supplies the result in `out_matrix`.

In C, `vec_$mat_multn` calculates the product of `matrix2` on the left and `matrix1` on the right, and the resulting operation is

```

for (i = 0; i < m; ++i)
  for (j = 0; j < s; ++j) {
    out_matrix[j][i] = 0.0;
    for (k = 0; k < n; ++k)
      out_matrix[j][i] += matrix1[k][i]
                        * matrix2[j][k];
  }

```

In Pascal, `vec $mat_multn` calculates the product of *matrix2* on the left and *matrix1* on the right, and the resulting operation is

```

for i := 1 to m do
  for j := 1 to s do
    begin
      out_matrix[j,i] = 0.0;
      for k := 1 to n do
        out_matrix[j,i] := out_matrix[j,i]
                          + matrix1[k,i]
                          * matrix2[j,k];
      end;
    end;

```

In FORTRAN, `vec $mat_multn` calculates the product of *matrix1* on the left and *matrix2* on the right, and the resulting operation is

```

do 10 i = 1, m
  do 10 j = 1, s
    out_matrix(i,j) = 0.0
    do 10 k = 1, n
      out_matrix(i,j) = out_matrix(i,j)
                        + matrix1(i,k)
                        * matrix2(k,j)
    &
    &
  10 continue

```

*matrix1* A matrix to be multiplied.

*matrix2* Another matrix to be multiplied.

*m, n, s* The various matrix dimensions.

*out\_matrix*

The product of *matrix1* and *matrix2*.

#### SEE ALSO

`vec_$dmat_multn`, `vec_$imat_multn`, `vec_$imat_multn16`.

## NAME

`vec_$max` – find the maximum absolute value in a single-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$max(
    float *vector,
    long int &length,
    float *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$max(
    in vector: univ vec_$real_vector;
    in length: integer32;
    out result: real;
    out location: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *vector*(*nvec*), *result*  
integer\*4 *length*, *location*

call `vec_$max`(*vector*, *length*, *result*, *location*)

## DESCRIPTION

`Vec_$max` searches through *length* elements of *vector*, and supplies the value and location of the element with the greatest absolute value.

In C, the resulting operation is

```

result = (float) fabs(vector[0]);
location = 1;
for (i = 1; i < length; ++i)
    if ((float) fabs(vector[i]) > result) {
        location = i + 1;
        result = (float) fabs(vector[i]);
    }

```

In Pascal, the resulting operation is

```

result := abs(vector[1]);
location = 1;
for 10 i := 2 to length do
    if (abs(vector[i]) > result) then
        begin
            location := i;
            result := abs(vector[i]);
        end

```

In FORTRAN, the resulting operation is

```

result = abs(vector(1))
location = 1
do 10 i = 2, length
    if (abs(vector(i)) .gt. result) then
        location = i
        result = abs(vector(i))
    endif
10 continue

```

*vector* The vector to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, *vec\_\$max* searches a row vector; whereas in FORTRAN, it searches a column vector.

VEC\_\$MAX

VEC\_\$MAX

**SEE ALSO**

vec\_\$dmax, vec\_\$imax, vec\_\$imax16, vec\_\$max\_i.

## NAME

`vec_$max_i` – find the maximum absolute value in a vector from a single-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$max_i(
    float *vector,
    long int &inc,
    long int &length,
    float *result,
    long int *location)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$max_i(
    in vector: univ_vec_$real_vector;
    in inc: integer32;
    in length: integer32;
    out result: real;
    out location: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *vector*(*nvec*), *result*  
integer\*4 *length*, *inc*, *location*

call `vec_$max_i`(*vector*, *inc*, *length*, *result*, *location*)

## DESCRIPTION

`Vec_$max_i` searches through the *length* elements of *vector* selected by *inc*, and supplies the value and location of the element with the greatest absolute value.

Through appropriate choice of *inc*, a program can use `vec_$max_i` to search a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

result = (float) fabs(vector[0]);
location = 1;
j = inc;
for (i = 1; i < length, ++i) {
    if ((float) fabs(vector[j]) > result) {
        location = i + 1;
        result = (float) fabs(vector[j]);
    }
    j += inc;
}

```

In Pascal, the resulting operation is

```

result := abs(vector[1]);
location := 1;
j := 1 + inc;
for 10 i := 2 to length do
    begin
        if (abs(vector[j]) > result) then
            begin
                location := i;
                result := abs(vector[j]);
            end
        j := j + inc;
    end

```

In FORTRAN, the resulting operation is

```

result = abs(vector(1))
location = 1
j = 1 + inc
do 10 i = 2, length
    if (abs(vector(j)) .gt. result) then
        location = i
        result = abs(vector(j))
    endif
    j = j + inc
10 continue

```

*vector* The array to search.

*inc* An increment for the index of *vector* that selects the elements to search.

*length* The number of elements to search.

*result* The maximum absolute value of all the elements searched.

*location* The location of the element with the greatest absolute value. The location supplied in *location* is just the index of the element with the greatest absolute value in FORTRAN or Pascal (if *vector* is declared to begin with index 1). In C, *location* - 1 is the index of the element.

#### NOTES

In C and Pascal, `vec_$max_i` searches a column vector; whereas in FORTRAN, it searches a row vector.

#### SEE ALSO

`vec_$dmax_i`, `vec_$imax16_i`, `vec_$imax_i`, `vec_$max`.

**NAME**

`vec_$mult_add` – scale and add one single-precision vector to another

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$mult_add(
    float *add_vec,
    float *mult_vec,
    long int &length,
    float &constant,
    float *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$mult_add(
    in add_vec: univ vec_$real_vector;
    in mult_vec: univ vec_$real_vector;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$real_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real add_vec(nvec), mult_vec(nvec), result_vec(nvec), constant
integer*4 length
```

```
call vec_$mult_add(add_vec, mult_vec, length,
& constant, result_vec)
```

**DESCRIPTION**

`Vec_$mult_add` multiplies the vector `mult_vec` by the scalar `constant`, adds the product to the vector `add_vec`, and supplies the resulting vector in `result_vec`.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = add_vec[i]
    + constant * mult_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
    result_vec[i] := add_vec[i]
                    + constant * mult_vec[i];

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
    result_vec(i) = add_vec(i)
&                + constant * mult_vec(i)
10 continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*length* The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

When *vec\_\$mult\_add* is used to operate on matrixes in C and Pascal, *add\_vec*, *mult\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_add*, *vec\_\$simult\_add*, *vec\_\$simult\_add16*, *vec\_\$mult\_add.i*.

**NAME**

`vec_$mult_add_i` – scale and add single-precision vectors in matrixes

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$mult_add_i(
    float *add_vec,
    long int &inc1,
    float *mult_vec,
    long int &inc2,
    long int &length,
    float &constant,
    float *result_vec,
    long int &inc3)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$mult_add_i
in add_vec: univ vec_$real_vector;
in inc1: integer32;
in mult_vec: univ vec_$real_vector;
in inc2: integer32;
in length: integer32;
in constant: real;
out result_vec: univ vec_$real_vector;
in inc3: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *add\_vec*(*nvec*), *mult\_vec*(*nvec*), *result\_vec*(*nvec*), *constant*  
integer\*4 *length*, *inc1*, *inc2*, *inc3*

call `vec_$mult_add_i`(*add\_vec*, *inc1*, *mult\_vec*, *inc2*, *length*,  
& *constant*, *result\_vec*, *inc3*)

**DESCRIPTION**

`Vec_$mult_add_i` multiplies *length* elements of *mult\_vec* selected by *inc2* by the scalar *constant*, adds the resulting products to elements of *add\_vec* selected by *inc1*, and supplies the results in elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use `vec_$mult_add_i` to scale and sum individual vectors in two matrixes and place the result in a vector of another matrix. To scale and add the *N*th vector in matrix *Y* to the *M*th vector in matrix *X*, choose *inc2* equal to the number of vectors in matrix *Y* and *inc1* equal to the number of vectors in matrix *X*. Then place the *M*th element of matrix *X* at the beginning of *add\_vec*, and place the *N*th element of matrix *Y* at the beginning of *mult\_vec*. To place the result of the operation in the *P*th vector of a resultant matrix, choose *inc3* equal to the number of vectors in the resultant matrix, and place the *P*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length; ++i) {
    result_vec[l] = add_vec[j]
                  + constant * mult_vec[k];
    j += inc1;
    k += inc2;
    l += inc3;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[l] := add_vec[j]
                  + constant * mult_vec[k];
    j := j + inc1;
    k := k + inc2;
    l := l + inc3;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      l = 1
      do 10 i = 1, length
          result(l) = add_vec(j)
&          + constant * mult_vec(k)
          j = j + inc1
          k = k + inc2
          l = l + inc3
10     continue

```

*add\_vec*

The vector to add to the product of *mult\_vec* and *constant*.

*inc1*

An increment for the index of *add\_vec* that selects the elements to add to the product of *mult\_vec* and *constant*.

*mult\_vec*

The vector to scale by *constant* and add to *add\_vec*.

*inc2*

An increment for the index of *mult\_vec* that selects the elements to multiply by *constant* and add to *add\_vec*.

*length*

The number of elements to use in the calculation.

*constant*

The scalar value used to scale *mult\_vec*.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

*inc3*

An increment for the index of *result\_vec* that selects the elements to receive the results of multiplying *mult\_vec* by *constant* and adding the product to *add\_vec*.

#### NOTES

In C and Pascal, *vec\_\$mult\_add\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

#### SEE ALSO

*vec\_\$dmult\_add\_i*, *vec\_\$imult\_add16\_i*, *vec\_\$imult\_add\_i*, *vec\_\$mult\_add*.

## NAME

`vec_$mult_constant` – multiply a single-precision vector by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$mult_constant(
    float *mult_vec,
    long int &length,
    float &constant,
    float *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$mult_constant(
    in mult_vec: univ vec_$real_vector;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$real_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *mult\_vec*(*nvec*), *result\_vec*(*nvec*), *constant*  
integer\*4 *length*

call `vec_$mult_constant`(*mult\_vec*, *length*, *constant*, *result\_vec*)

## DESCRIPTION

`Vec_$mult_constant` multiplies the single-precision array *mult\_vec* by the scalar value *constant* and supplies the result in the array *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = constant * start_vec[i];
```

In Pascal, the resulting operation is

```

for i := 1 to length do
  begin
    result_vec[i] := constant * start_vec[i];
  end

```

In FORTRAN, the resulting operation is

```

do 10 i = 1, length
  result_vec(i) = constant * mult_vec(i)
10 continue

```

*mult\_vec*

The vector to multiply by *constant*.

*length* The number of elements to multiply.

*constant*

The scalar constant to multiply *mult\_vec* by.

*result\_vec*

The vector resulting from multiplying *mult\_vec* by *constant*.

#### NOTES

When *vec\_\$mult\_constant* is used to operate on matrixes in C and Pascal, *mult\_vec* and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dmult\_constant*,  
*vec\_\$mult\_constant\_i*.

*vec\_\$imult\_constant*,

*vec\_\$imult\_constant16*,

## NAME

`vec_$mult_constant_i` – multiply a vector in a single-precision matrix by a scalar

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$mult_constant_i(
    float *mult_vec,
    long int &incl,
    long int &length,
    float &constant,
    float *result_vec,
    long int &inc2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$mult_constant_i(
    in mult_vec: univ vec_$real_vector;
    in incl: integer32;
    in length: integer32;
    in constant: real;
    out result_vec: univ vec_$real_vector;
    in inc2: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *mult\_vec*(*nvec*), *result\_vec*(*nvec*), *constant*  
integer\*4 *length*, *incl*, *inc2*

call `vec_$mult_constant_i`(*mult\_vec*, *incl*, *length*, *constant*,  
& *result\_vec*, *inc2*)

## DESCRIPTION

`Vec_$mult_constant_i` multiplies *length* elements of the single-precision array *mult\_vec* selected by *incl* by the scalar value *constant*, and supplies the result in elements of the array *result\_vec* selected by *inc2*.

Through appropriate choice of *incl* and *inc2*, a program can use `vec_$mult_constant_i` to operate on a vector in a matrix. To multiply *M*th vector in a matrix by *constant*, choose *incl* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *mult\_vec*. To place the result of the operation in the

*N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *result\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    result_vec[k] = constant * mult_vec[j];
    k += inc2;
    j += incl;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    result_vec[k] := constant * mult_vec[j];
    k := k + inc2;
    j := j + incl;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          result(j) = constant * mult_vec(k)
          j = j + inc2
          k = k + incl
10    continue

```

*mult\_vec*

The vector to multiply by *constant*.

*incl*

An increment for the index of the array *mult\_vec* that selects elements to multiply by *constant*.

*length*

The number of products to calculate.

*constant*

The scalar constant to multiply elements of *mult\_vec* by.

*result\_vec*

An array whose elements receive the product of *constant* and *mult\_vec*.

VEC\_\$MULT\_CONSTANT\_I

VEC\_\$MULT\_CONSTANT\_I

*inc2* An increment for the index of the array *result\_vec* that selects elements to receive the product of *constant* and *mult\_vec*.

**NOTES**

In C and Pascal, *vec\_\$mult\_constant\_i* operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

*vec\_\$dmult\_constant\_i*, *vec\_\$imult\_constant16\_i*, *vec\_\$imult\_constant\_i*,  
*vec\_\$mult\_constant*.

## NAME

`vec_$postmult` – multiply a single-precision vector by a 4x4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$postmult(
    float *matrix,
    float *start_vec,
    float *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$postmult
in matrix: univ vec_$real_matrix;
in start_vec: univ vec_$real_vector;
out result_vec: univ vec_$real_vector;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
real matrix(4, 4), start_vec(4), result_vec(4)
```

```
call vec_$postmult(matrix, start_vec, result_vec)
```

## DESCRIPTION

`vec_$postmult` multiplies the 4-element vector `start_vec` by the 4x4 matrix `matrix`.

In C, `vec_$postmult` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```
for (j = 0; j < 4; ++j) {
    result_vec[j] = 0.0;
    for (i = 0; i < 4; ++i)
        result_vec[j] += start_vec[i]
            * matrix[i][j];
}
```

In Pascal, `vec_$postmult` applies `matrix` as a right transform to a row vector `start_vec`, and the resulting operation is

```

for j := 1 to 4 do
begin
result_vec[j] := 0.0;
for i := 1 to 4 do
    result_vec[j] := result_vec[j]
                    + start_vec[i]
                    * matrix[i,j];
end

```

In FORTRAN, `vec_$postmult` applies *matrix* as a left transform to a column vector *start\_vec*, and the resulting operation is

```

do 10 j = 1, 4
    result_vec(j) = 0.0
    do 10 i = 1, 4
        result_vec(j) = result_vec(j)
                        + start_vec(i)
                        * matrix(j,i)
    &
    &
10 continue

```

*matrix* The matrix to multiply by *start\_vec*.

*start\_vec* The vector to multiply by *matrix*.

*result\_vec* The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$premult` transforms single-precision vectors from the other side.

#### SEE ALSO

`vec_$dpostmult`, `vec_$ipostmult`, `vec_$postmult16`.

**NAME**

`vec_$postmultn` – multiply a single-precision vector by a matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$postmultn(
    float *matrix,
    float *start_vec,
    long int &m,
    long int &n,
    float *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
#include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$postmultn(
    in matrix: univ vec_$real_matrix;
    in start_vec: univ vec_$real_vector;
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$real_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
#include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
real matrix(m, n), start_vec(n), result_vec(m)
```

```
call vec_$postmultn(matrix, start_vec, m, n, result_vec)
```

**DESCRIPTION**

`Vec_$postmultn` multiplies the  $n$ -element vector `start_vec` by the variably dimensioned matrix `matrix`, and supplies the resulting  $m$ -element vector in `result_vec`.

In C, `vec_$postmultn` applies the  $n \times m$  matrix `matrix` as a right transform to the  $m$ -element row vector `start_vec`, and supplies the transformed  $n$ -element result in `result_vec`:

```

for (i = 0; i < m; ++i) {
    result_vec[i] = 0.0;
    for (j = 0; j < n; ++j)
        result_vec[i] += start_vec[j]
                        * matrix[j][i];
}

```

In Pascal, `vec_$postmultn` applies the  $n \times m$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to m do
begin
    result_vec[i] := 0.0;
    for j := 1 to n do
        result_vec[i] := result_vec[i]
                        + start_vec[j]
                        * matrix[j,i];
    end;
end;

```

In FORTRAN, `vec_$postmultn` applies the  $m \times n$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, m
    result_vec(i) = 0.0
    do 10 j = 1, n
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(i,j)
    &
    &
10 continue

```

*matrix* A matrix to multiply *start\_vec* by.

*start\_vec* An  $n$ -element vector to multiply by *matrix*.

*m* The number of elements in *start\_vec*.

*n* The number of elements in *result\_vec*.

*result\_vec* An  $m$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$premultn` transforms single-precision vectors from the other side.

#### SEE ALSO

`vec_$dpostmultn`, `vec_$postmultn`, `vec_$postmultn16`.

## NAME

`vec_$premult` – multiply a single-precision vector by a 4×4 matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$premult(
    float *start_vec,
    float *matrix,
    float *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$premult(
    in start_vec: univ vec_$real_vector;
    in matrix: univ vec_$real_matrix;
    out result_vec: univ vec_$real_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
real start_vec(4), matrix(4,4), result_vec(4)
```

```
call vec_$premult(start_vec, matrix, result_vec)
```

## DESCRIPTION

`Vec_$premult` multiplies the 4-element vector `start_vec` by the 4×4 matrix `matrix`.

In C, `vec_$premult` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```
for (i = 0; i < 4; ++i) {
    result_vec[i] = 0.0;
    for (j = 0; j < 4; ++j)
        result_vec[i] += start_vec[j]
            * matrix[i][j];
}
```

In Pascal, `vec_$premult` applies `matrix` as a left transform to a column vector `start_vec`, and the resulting operation is

```

for i := 1 to 4 do
begin
result_vec[i] := 0.0;
for j := 1 to 4 do
    result_vec[i] := result_vec[i]
                    + start_vec[i]
                    * matrix[i,j];
end

```

In FORTRAN, `vec_$premult` applies *matrix* as a right transform to a row vector *start\_vec*, and the resulting operation is

```

do 10 i = 1, 4
    result_vec(i) = 0.0
    do 10 j = 1, 4
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(j,i)
10 continue

```

*start\_vec*

The vector to multiply by *matrix*.

*matrix* The matrix to multiply by *start\_vec*.

*result\_vec*

The product of *start\_vec* and *matrix*.

#### NOTES

`Vec_$postmult` transforms single-precision vectors from the other side.

#### SEE ALSO

`vec_$dpremult`, `vec_$ipremult`, `vec_$ipremult16`.

**NAME**

`vec_$premultn` – multiply a single-precision vector by a matrix

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$premultn(
    float *start_vec,
    float *matrix,
    long int &m,
    long int &n,
    float *result_vec)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
#include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$premultn(
    in start_vec: univ vec_$real_vector;
    in matrix: univ vec_$real_matrix;
    in m: integer32;
    in n: integer32;
    out result_vec: univ vec_$real_vector);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
#include '/sys/ins/vec.ins.ftn'
```

```
integer*4 m, n
parameter (m = 3, n = 4)
```

```
real start_vec(m), matrix(m, n), result_vec(n)
```

```
call vec_$premultn(start_vec, matrix, m, n, result_vec)
```

**DESCRIPTION**

`Vec_$premultn` multiplies the  $m$ -element vector `start_vec` by the variably dimensioned matrix `matrix`, and supplies the resulting  $n$ -element vector in `result_vec`.

In C, `vec_$premultn` applies the  $n \times m$  matrix `matrix` as a left transform to the  $m$ -element column vector `start_vec`, and supplies the transformed  $n$ -element result in `result_vec`:

```

for (i = 0; i < n; ++i) {
    result_vec[i] = 0.0;
    for (j = 0; j < m; ++j)
        result_vec[i] += start_vec[j]
                        * matrix[i][j];
}

```

In Pascal, `vec_$premultn` applies the  $n \times m$  matrix *matrix* as a left transform to the  $m$ -element column vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

for i := 1 to n do
begin
    result_vec[i] := 0.0;
    for j := 1 to m do
        result_vec[i] = result_vec[i]
                        + start_vec[j]
                        * matrix[i,j];
    end
end

```

In FORTRAN, `vec_$premultn` applies the  $m \times n$  matrix *matrix* as a right transform to the  $m$ -element row vector *start\_vec*, and supplies the transformed  $n$ -element result in *result\_vec*:

```

do 10 i = 1, n
    result_vec(i) = 0.0
    do 10 j = 1, m
        result_vec(i) = result_vec(i)
                        + start_vec(j)
                        * matrix(j,i)
    &
    &
10 continue

```

*start\_vec*

An  $m$ -element vector to multiply by *matrix*.

*matrix* A matrix to multiply *start\_vec* by.

$m$  The number of elements in *start\_vec*.

$n$  The number of elements in *result\_vec*.

*result\_vec*

An  $n$ -element vector that is the product of *matrix* and *start\_vec*.

#### NOTES

`Vec_$postmultn` transforms single-precision vectors from the other side.

#### SEE ALSO

`vec_$dpremultn`, `vec_$ipremultn`, `vec_$ipremultn16`.

**NAME**

`vec_$sp_dp` – copy a single-precision vector to a double-precision vector

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$sp_dp(
    float *sp_vec,
    double *dp_vec,
    long int &length)
```

**SYNOPSIS (Pascal)**

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$sp_dp(
    in sp_vec: univ vec_$real_vector;
    in dp_vec: univ vec_$double_vector;
    in length: integer32);
```

**SYNOPSIS (FORTRAN)**

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
double precision dp_vec(nvec)
real sp_vec(nvec)
integer*4 length
```

```
call vec_$sp_dp(sp_vec, dp_vec, length)
```

**DESCRIPTION**

`Vec_$sp_dp` copies *length* elements from the single-precision vector *sp\_vec* to the double-precision vector *dp\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    dp_vec[i] = (double)sp_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
  begin
    dp_vec[i] := sp_vec[i];
  end
```

In FORTRAN, the resulting operation is

```
do 10 i=1, length
  dp_vec(i) = dble(sp_vec(i))
10 continue
```

*sp\_vec* The single-precision vector to copy from.  
*dp\_vec* The double-precision vector to copy to.  
*length* The number of elements to copy.

#### NOTES

In C and Pascal, `vec_$sp_dp` copies row vectors; whereas in FORTRAN, it copies column vectors.

#### SEE ALSO

`vec_$dp_sp`, `vec_$sp_dp_i`.

## NAME

`vec_$sp_dp_i` – copy a single-precision vector to a double-precision vector in matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$sp_dp_i(
    float *sp_vec,
    long int &inc1,
    double *dp_vec,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$sp_dp_i(
    in sp_vec: univ vec_$real_vector;
    in inc1: integer32;
    in dp_vec: univ vec_$double_vector;
    in inc2: integer32;
    in length: integer32) ;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

double precision *dp\_vec*(*nvec*)  
 real *sp\_vec*(*nvec*)  
 integer\*4 *length*, *inc1*, *inc2*

call `vec_$sp_dp_i`(*sp\_vec*, *inc1*, *dp\_vec*, *inc2*, *length*)

## DESCRIPTION

`Vec_$sp_dp_i` copies *length* elements selected by *inc1* from the single-precision array *sp\_vec* to elements of the double-precision array *dp\_vec* chosen by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$sp_dp_i` to copy a vector from one matrix to another. To copy the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *sp\_vec*. To place the copy into the *N*th vector of a matrix, choose *inc2* equal to the number of vectors in the resultant matrix, and place the *N*th element of the matrix array at the beginning of *dp\_vec*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length; ++i) {
    dp_vec[i] = (double)sp_vec[i];
    j += inc1;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
begin
    dp_vec[i] := sp_vec[i];
    j := j + inc1;
    k := k + inc2;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          dp_vec(k) = dble(sp_vec(j))
          j = j + inc1
          k = k + inc2
10    continue

```

*sp\_vec* The single-precision array to copy from.

*inc1* Increment for the index of *sp\_vec* that selects the elements to copy from.

*dp\_vec* The double-precision array to copy to.

*inc2* Increment for the index of *dp\_vec* that selects the elements to copy to.

*length* The number of elements to copy from *sp\_vec* to *dp\_vec*.

#### NOTES

In C and Pascal, `vec_$sp_dp_i` copies column vectors; whereas in FORTRAN, it copies row vectors.

#### SEE ALSO

`vec_$dp_sp_i`, `vec_$sp_dp`.

## NAME

`vec_$sub` – subtract single-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$sub(
    float *start_vec,
    float *sub_vec,
    long int &length,
    float *result_vec)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$sub(
    in start_vec: univ vec_$real_vector;
    in sub_vec: univ vec_$real_vector;
    in length: integer32;
    out result_vec: univ vec_$real_vector);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real start_vec(nvec), sub_vec(nvec), result_vec(nvec)
integer*4 length
```

call `vec_$sub(start_vec, sub_vec, length, result_vec)`

## DESCRIPTION

`Vec_$sub` subtracts *length* elements of *sub\_vec* from *start\_vec*, and supplies the difference in *result\_vec*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    result_vec[i] = start_vec[i] - sub_vec[i];
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    result_vec[i] := start_vec[i] - sub_vec[i];
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    result_vec(i) = start_vec(i) - sub_vec(i)
10 continue
```

*start\_vec*

The vector to subtract *sub\_vec* from.

*sub\_vec* The vector to subtract from *start\_vec*.

*length* The number of differences to calculate.

*result\_vec*

The difference of *start\_vec* and *sub\_vec*.

#### NOTES

When *vec\_\$sub* is used to operate on matrixes in C and Pascal, *start\_vec*, *result\_vec*, and *result\_vec* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

*vec\_\$dsub*, *vec\_\$isub*, *vec\_\$isub16*, *vec\_\$sub.i*.

## NAME

`vec_$sub_i` – subtract single-precision vectors in matrixes

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$sub_i(
    float *start_vec,
    long int &inc1,
    float *sub_vec,
    long int &inc2,
    long int &length,
    float *result_vec,
    long int &inc3)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$sub_i(
    in start_vec: univ vec_$real_vector;
    in inc1: integer32;
    in sub_vec: univ vec_$real_vector;
    in inc2: integer32;
    in length: integer32;
    out result_vec: univ vec_$real_vector;
    in inc3: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real start_vec(nvec), sub_vec(nvec), result_vec(nvec)
integer*4 length, inc1, inc2, inc3
```

```
call vec_$sub_i(start_vec, inc1, sub_vec, inc2,
& length, result_vec, inc3)
```

## DESCRIPTION

`Vec_$sub_i` subtracts *length* elements of *sub\_vec* selected by *inc2* from *length* elements of *start\_vec* selected by *inc1*, and supplies the result in the elements of *result\_vec* selected by *inc3*.

Through appropriate choice of *inc1*, *inc2*, and *inc3*, a program can use `vec_$sub_i` to subtract individual vectors in two matrixes and place the difference in a vector of another

matrix. To subtract the  $N$ th vector in matrix  $Y$  from the  $M$ th vector in matrix  $X$ , choose  $inc2$  equal to the number of vectors in matrix  $Y$  and  $inc1$  equal to the number of vectors in matrix  $X$ . Then place the  $M$ th element of matrix  $X$  at the beginning of  $start\_vec$ , and place the  $N$ th element of matrix  $Y$  at the beginning of  $sub\_vec$ . To place the result of the operation in the  $P$ th vector of a resultant matrix, choose  $inc3$  equal to the number of vectors in the resultant matrix, and place the  $P$ th element of the matrix array at the beginning of  $result\_vec$ .

In C, the resulting operation is

```

j = 0;
k = 0;
l = 0;
for (i = 0; i < length, ++i) {
    result_vec[j] = start_vec[k] - sub_vec[l];
    j += inc3;
    k += inc1;
    l += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
l := 1;
for i := 1 to length do
begin
    result_vec[j] := start_vec[k] - sub_vec[l];
    j := j + inc3;
    k := k + inc1;
    l := l + inc2;
end

```

In FORTRAN, the resulting operation is

```

j = 1
k = 1
l = 1
do 10 i = 1, length
    result_vec(j) = start_vec(k) - sub_vec(l)
    j = j + inc3
    k = k + inc1
    l = l + inc2
10 continue

```

- start\_vec* The vector to subtract *sub\_vec* from.
- inc1* An increment for the index of *start\_vec* that chooses which elements the elements of *sub\_vec* will be subtracted from.
- sub\_vec* The vector to subtract from *start\_vec*.
- inc2* An increment for the index of *sub\_vec* that chooses which elements will be subtracted from the elements of *start\_vec*.
- length* The number of scalar differences to calculate.
- result\_vec* The difference of *start\_vec* and *sub\_vec*.
- inc3* An increment for the index of *result\_vec* that chooses which elements will received the differences.

**NOTES**

In C and Pascal, `vec_$sub_i` operates on column vectors; whereas in FORTRAN, it operates on row vectors.

**SEE ALSO**

`vec_$dsub_i`, `vec_$isub16_i`, `vec_$isub_i`, `vec_$sub`.

## NAME

vec\_\$sum – sum the elements of a single-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
float vec_$sum(
    float *vec,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$sum(
    in vec: univ vec_$real_vector;
    in length: integer32): real;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

```
real vec(nvec), sum
integer*4 length
```

```
sum = vec_$sum(vec, length)
```

## DESCRIPTION

Vec\_\$sum adds together *length* elements of the single-precision array *vector* and returns the sum.

In C, the resulting operation is

```
return_value = 0.0;
for (i = 0; i < length; ++i)
    return_value += vec[i];
```

In Pascal, the resulting operation is

```
return_value := 0.0;
for i := 1 to length do
    return_value := return_value + vec[i];
```

In FORTRAN, the resulting operation is

```
      vec_$sum = 0.0
      do 10 i = 1, length
          vec_$sum = vec_$sum + vec(i)
10    continue
```

*vec*     The vector to sum.

*length*   The number of elements in *vec* to sum.

#### NOTES

When **vec\_\$sum** is used to operate on matrixes in C and Pascal, *vec* is a row vector; whereas in FORTRAN, it is a column vector.

#### SEE ALSO

vec\_\$dsum, vec\_\$isum, vec\_\$isum16, vec\_\$sum\_i.

## NAME

`vec_$sum_i` – sum the elements of a vector in a single-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
float vec_$sum_i(
    float *vec,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
function vec_$sum_i(
    in vec: univ vec_$real_vector;
    in inc: integer32;
    in length: integer32): real;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

parameter (*nvec* = 10)

real *vec*(*nvec*), *sum*  
integer\*4 *length*, *inc*

*sum* = `vec_$sum_i`(*vec*, *inc*, *length*)

## DESCRIPTION

`Vec_$sum_i` adds *length* elements from the single-precision array *vector* selected by *inc* and returns the sum.

Through appropriate choice of *inc*, a program can use `vec_$sum_i` to sum an individual vector in a matrix. To sum the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix.

In C, the resulting operation is

```
return_value = 0.0;
j = 0;
for (i = 0; i < length; ++i) {
    return_value += vec[j];
    j += inc;
}
```

In Pascal, the resulting operation is

```
return_value := 0.0;
j := 1;
for i := 1 to length do
    begin
        return_value := return_value + vec[j];
        j := j + inc;
    end
```

In FORTRAN, the resulting operation is

```
vec_$sum_i = 0.0
j = 1
do 10 i = 1, length
    vec_$sum_i = vec_$sum_i + vec(j)
    j = j + inc
10 continue
```

*vec* An array that contains the elements to sum.

*inc* An increment for the index of *vec* that selects which elements of *vec* are summed.

*length* The number of elements in *vec* to sum.

#### NOTES

In C and Pascal, *vec\_\$sum\_i* sums a column vector; whereas in FORTRAN, it sums a row vector.

#### SEE ALSO

*vec\_\$dsum\_i*, *vec\_\$isum16\_i*, *vec\_\$isum\_i*, *vec\_\$sum*.

## NAME

`vec_$swap` – swap two single-precision vectors

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$swap(
    float *vec1,
    float *vec2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$swap(
    var vec1: univ vec_$real_vector;
    var vec2: univ vec_$real_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real vec1(nvec), vec2(nvec)
integer*4 length
```

```
call vec_$swap(vec1, vec2, length)
```

## DESCRIPTION

`Vec_$swap` swaps *length* elements between the single-precision vectors *vec1* and *vec2*.

In C, the resulting operation is

```
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
}
```

In Pascal, the resulting operation is

```
for i := 1 to length do
  begin
    temp := vec1[i];
    vec1[i] := vec2[i];
    vec2[i] := temp;
  end
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
  temp = vec1(i)
  vec1(i) = vec2(i)
  vec2(i) = temp
10 continue
```

*vec1* The vector to be swapped with *vec2*.  
*vec2* The vector to be swapped with *vec1*.  
*length* The number of elements to swap.

#### NOTES

When `vec_$swap` is used to operate on matrixes in C and Pascal, *vec1* and *vec2* are row vectors; whereas in FORTRAN, they are column vectors.

#### SEE ALSO

`vec_$dswap`, `vec_$iswap`, `vec_$iswap16`, `vec_$swap_i`.

## NAME

`vec_$swap_i` – swap two vectors in a single-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$swap_i(
    float *vec1,
    long int &inc1,
    float *vec2,
    long int &inc2,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$swap_i(
    var vec1: univ vec_$real_vector;
    in inc1: integer32;
    var vec2: univ vec_$real_vector;
    in inc2: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real vec1(nvec), vec2(nvec)
integer*4 length, inc1, inc2
```

```
call vec_$swap_i(vec1, inc1, vec2, inc2, length)
```

## DESCRIPTION

`Vec_$swap_i` swaps the *length* elements of *vec1* selected by *inc1* with the *length* elements of *vec2* selected by *inc2*.

Through appropriate choice of *inc1* and *inc2*, a program can use `vec_$swap_i` to swap vectors between two matrixes. To select the *M*th vector in a matrix, choose *inc1* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vec1*. To swap the selected vector with the *N*th vector of the same or another matrix, choose *inc2* equal to the number of vectors in the same or other matrix, and place the *N*th element of the matrix array at the beginning of *vec2*.

In C, the resulting operation is

```

j = 0;
k = 0;
for (i = 0; i < length, ++i) {
    temp = vec1[i];
    vec1[i] = vec2[i];
    vec2[i] = temp;
    j += incl;
    k += inc2;
}

```

In Pascal, the resulting operation is

```

j := 1;
k := 1;
for i := 1 to length do
    begin
        temp := vec1[i];
        vec1[i] := vec2[i];
        vec2[i] := temp;
        j := j + incl;
        k := k + inc2;
    end

```

In FORTRAN, the resulting operation is

```

      j = 1
      k = 1
      do 10 i = 1, length
          temp = vec1(j)
          vec1(j) = vec2(k)
          vec2(k) = temp
          j = j + incl
          k = k + inc2
10    continue

```

*vec1* The vector to be swapped with *vec2*.

*incl* The increment for the index of *vec1* that selects the elements to be swapped with *vec2*.

*vec2* The vector to be swapped with *vec1*.

*inc2* The increment for the index of *vec2* that selects the elements to be swapped with *vec1*.

*length* The number of elements to swap.

**NOTES**

In C and Pascal, `vec_$swap_i` swaps column vectors; whereas in FORTRAN, it swaps row vectors.

**SEE ALSO**

`vec_$dswap_i`, `vec_$iswap16_i`, `vec_$iswap_i`, `vec_$swap`.

## NAME

`vec_$zero` – zero a single-precision vector

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$zero(
    float *vector,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$zero(
    var vector: univ vec_$real_vector;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real vector(nvec)
integer*4 length
```

```
call vec_$zero(vector, length)
```

## DESCRIPTION

`Vec_$zero` zeros the first *length* elements of the single-precision vector *vector*.

In C, the resulting operation is

```
for (i = 0; i < length; ++i)
    vector[i] = 0.0;
```

In Pascal, the resulting operation is

```
for i := 1 to length do
    vector[i] := 0.0;
```

In FORTRAN, the resulting operation is

```
do 10 i = 1, length
    vec(i) = 0.0
10 continue
```

*vector* The vector to be zeroed.

*length* The number of elements in *vector* to zero.

**NOTES**

In C and Pascal, *vec\_\$zero* zeros row vectors; whereas in FORTRAN, it zeros column vectors.

**SEE ALSO**

*vec\_\$dzero*, *vec\_\$izero*, *vec\_\$izerol6*, *vec\_\$zero\_i*.

## NAME

`vec_$zero_i` – zero a vector in a single-precision matrix

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vec.h>
```

```
void vec_$zero_i(
    float *vector,
    long int &inc,
    long int &length)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vec.ins.pas';
```

```
procedure vec_$zero_i(
    var vector: univ vec_$real_vector;
    in inc: integer32;
    in length: integer32);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vec.ins.ftn'
```

```
parameter (nvec = 10)
```

```
real vector(nvec)
integer*4 length, inc
```

```
call vec_$zero_i(vector, inc, length)
```

## DESCRIPTION

`Vec_$zero_i` zeros the *length* elements of the single-precision array *vector* selected by *inc*.

Through appropriate choice of *inc*, a program can use `vec_$zero_i` to zero a vector within a matrix. To search the *M*th vector in a matrix, choose *inc* equal to the number of vectors in the matrix, and place the *M*th element of the matrix array at the beginning of *vector*.

In C, the resulting operation is

```

j = 0;
for (i = 0; i < length; ++i) {
    vector[i] = 0.0;
    j += inc;
}

```

In Pascal, the resulting operation is

```

j := 1;
for i := 1 to length do
begin
    vector[i] := 0.0;
    j := j + inc;
end

```

In FORTRAN, the resulting operation is

```

      j = 1
      do 10 i = 1, length
          vec(j) = 0.0
          j = j + inc
10    continue

```

*vector* The vector to be zeroed.

*inc* An increment for the index of *vector* that chooses the elements to be zeroed.

*length* The number of elements in *vector* to zero.

#### NOTES

In C and Pascal, `vec_$zero_i` zeros column vectors; whereas in FORTRAN, it zeros row vectors.

#### SEE ALSO

`vec_$dzero_i`, `vec_$zero16_i`, `vec_$zero_i`, `vec_$zero`.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses, income, and any other financial activities that occur within the organization.

Secondly, the document highlights the need for regular reconciliation of accounts. By comparing the internal records with bank statements and other external sources, discrepancies can be identified and corrected promptly. This process helps to prevent errors from accumulating and ensures that the books are balanced at all times.

Another key aspect mentioned is the importance of having a clear and consistent accounting policy. This policy should define the methods used for recording transactions, recognizing revenue, and measuring expenses. It should also specify the frequency of reporting and the level of detail required. A well-defined policy helps to ensure that all transactions are recorded in a uniform and transparent manner.

Finally, the document stresses the importance of maintaining confidentiality and security of financial information. This information is often sensitive and can be used to the advantage of competitors or other parties if it falls into the wrong hands. Therefore, it is essential to implement robust security measures, such as access controls, encryption, and secure storage, to protect the data from unauthorized access and theft.

---

**Contents**

intro .....	VFMT-1
vfmt_\$decode10 .....	VFMT-8
vfmt_\$decode2 .....	VFMT-10
vfmt_\$decode5 .....	VFMT-12
vfmt_\$encode .....	VFMT-14
vfmt_\$encode10 .....	VFMT-16
vfmt_\$encode2 .....	VFMT-18
vfmt_\$encode5 .....	VFMT-20
vfmt_\$read10 .....	VFMT-22
vfmt_\$read2 .....	VFMT-24
vfmt_\$read5 .....	VFMT-26
vfmt_\$rs10 .....	VFMT-28
vfmt_\$rs2 .....	VFMT-30
vfmt_\$rs5 .....	VFMT-32
vfmt_\$write .....	VFMT-34
vfmt_\$write10 .....	VFMT-36
vfmt_\$write2 .....	VFMT-38
vfmt_\$write5 .....	VFMT-39
vfmt_\$ws .....	VFMT-40
vfmt_\$ws10 .....	VFMT-42
vfmt_\$ws2 .....	VFMT-44
vfmt_\$ws5 .....	VFMT-46

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial statements. This includes not only sales and purchases but also expenses, income, and transfers between accounts.

The second part of the document provides a detailed explanation of the accounting cycle. It outlines the ten steps involved in the process, from identifying the accounting entity to preparing financial statements. Each step is described in detail, with examples provided to illustrate the concepts.

The third part of the document discusses the various types of accounts used in accounting. It explains the difference between assets, liabilities, and equity accounts, and how they are classified. It also discusses the importance of understanding the normal balances for each type of account.

The fourth part of the document discusses the process of adjusting entries. It explains why adjusting entries are necessary and how they are prepared. It provides examples of common adjusting entries, such as depreciation, amortization, and accruals.

The fifth part of the document discusses the preparation of financial statements. It explains how the adjusted trial balance is used to prepare the income statement, balance sheet, and statement of owner's equity. It also discusses the importance of reviewing the financial statements for accuracy and consistency.

The sixth part of the document discusses the closing process. It explains how the temporary accounts (revenues, expenses, and dividends) are closed to the permanent accounts (retained earnings). It provides a step-by-step guide to the closing process.

The seventh part of the document discusses the importance of internal controls. It explains how internal controls help to prevent errors and fraud, and how they can be designed to improve the efficiency of the accounting system.

The eighth part of the document discusses the role of the accountant. It explains the various responsibilities of an accountant, including recording transactions, preparing financial statements, and providing financial advice to management.

The ninth part of the document discusses the future of accounting. It discusses the impact of technology on the accounting profession and the need for accountants to stay current in their skills and knowledge.

The tenth part of the document discusses the importance of ethics in accounting. It explains how accountants should maintain the highest standards of ethical behavior and how they can avoid conflicts of interest.

**NAME**

intro – variable formatting package

**SYNOPSIS (C)**

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

**SYNOPSIS (Pascal)**

```
%include 'sys/ins/base.ins.pas';
%include 'sys/ins/vfmt.ins.pas';
```

**SYNOPSIS (FORTRAN)**

```
%include 'sys/ins/base.ins.ftn'
%include 'sys/ins/vfmt.ins.ftn'
```

**DESCRIPTION**

The `vfmt_$` calls facilitate formatted I/O, and arithmetic-to-ASCII conversions. They are most useful in Pascal or FORTRAN programs because their functionality largely overlaps that of C's `stdio` package.

**Control Strings**

All `vfmt_$` calls take a control string argument. The control string defines fields in an ASCII string that correspond to source or destination variables also provided in the call. Each `vfmt_$` call either writes data from source variables into fields of an ASCII string defined by the control string, or reads ASCII data fields defined by the control string into destination variables. The control string defines the types and the sizes of the data fields in the ASCII string, which must be compatible with the types of the source or destination variables. Control strings are made up of "format directives" that begin with a percent sign (%) and end with some terminating character defining their function. The `vfmt_$` calls ignore case in their control strings.

**Data Directives**

Each data field is defined by a single format directive in the control string. The `vfmt_$` calls recognize six different data formats: ASCII (%A), decimal (%D), scientific (%E), floating (%F), hexadecimal (%H), and octal (%O). Each basic data directive can take options between the percent sign and the terminating format character that modify the field definition. The following table summarizes the options appropriate for each data directive.

Data-Related Format Directives	
[Options] Directive	Function
% [fw][M length][E][Z][K][U L] A	encode/decode ASCII
% [fw][E][Z][J][U S P][W L] D	encode/decode integer decimal
% [fw][fw.dr][Z][J][S P][W L] E	encode scientific floating
% [fw][fw.dr][E][Z][J][S P][W L] F	encode/decode floating point
% [fw][E][Z][J][U S P][W L][Y] H	encode/decode integer hexadecimal
% [fw][E][Z][J][U S P][W L] O	encode/decode integer octal

### ASCII Data (%A)

The %A directive is used to define ASCII fields, and can be used in control strings with both read and write calls.

The options have the following effects on ASCII data:

*fw*      *Fw* is an integer between 1 and 65536 inclusive, indicating the width of the field. A field width option means read or write exactly this many characters, padding with blanks if necessary, unless overridden by the Z, E, or M options. If no field width is specified, a *vfmt\_\$* call will read or write only non-blank characters.

#### *M length*

*Length* is an integer between 1 and 65536 inclusive, indicating the number of significant bytes in a source variable, or the number of bytes provided in a destination buffer. The M option can be used to specify the number of bytes read from a source variable or written to a destination variable. If the M option is not used and there is no *fw* specifier, a read call will look at the value passed in a destination variable allocated to receive a string length.

- E      The E option is only valid in read calls, where it forces early termination of a data field when a delimiter is encountered. The default delimiters are blank and comma, but other delimiters can be declared with the %"..." directive.
- K      The K option is only valid in read calls, where it means ignore leading spaces in the input field; that is, spaces that precede the first visible character. K overrides E.
- Z      The Z option means include trailing spaces in the data field; that is, spaces that follow the last visible character. The Z option causes write calls to include trailing spaces from the source variable in the output. The Z option causes read calls to include trailing spaces from the input field when filling the destination variable. Without Z, trailing blanks are ignored.

- L The L option means convert all characters read or written to lowercase. It is not valid with the U option.
- U The U option means convert all characters read or written to uppercase. It is not valid with the L option.

#### Floating-Point Data (%E and %F)

The %E and %F directives are used to define floating-point fields. The %E directive is only valid in write calls, but the %F directive can be used in any control string.

The options have the following effects on floating-point data:

- fw* *Fw* is an integer between 1 and 100 inclusive, indicating the total width of the data field including integer and fractional parts of the mantissa, the sign, decimal point, and exponent (if any). In a write call, if the field width specified is too small to contain the value to be written, the field will be filled with asterisks (\*) to indicate an overflow condition. If no field width is specified, a write call uses a field just large enough to hold the value, and a read call stops at the first field delimiter it encounters.
- dw* *Dw* is an integer specifying the width of the fractional part of the mantissa; that is, the number of digits to the right of the decimal point. It is only valid in write calls, and the default is 2.
- E The E option is only valid in read calls, where it forces early termination of a data field when a delimiter is encountered. The default delimiters are blank and comma, but other delimiters can be declared with the %"... " directive. The E option overrides the field width specifier.
- Z The Z option is only valid in write calls, where it means add zeros (0) to the left of the number to fill the output data field. Z is not valid without a field width specifier (*fw*).
- J The J option is only valid in write calls, where it means left-justify the number in the output data field. Without the J option, numbers are right justified in the field.
- S The S option means the number to be read or written has a minus sign if it is negative, and no sign if it is positive. This is the default unless the P option is specified. S is always redundant for read calls because ASCII data must be signed if negative.
- P The P option is only valid in write calls, and means prepend a minus sign to the number if it is negative, and prepend a plus sign if it is positive.
- W The W option means the number to be read or written is a single-precision floating-point value. If W is not specified, L is the default.
- L The L option means the number to be read or written is a double-precision floating-point value. It is not a valid option if W is specified.

**Integer Data (%D, %H, and %O)**

The %D, %H, and %O directives are used to define integer fields. They can be used in both read and write calls.

- fw*     *Fw* is an integer between 1 and 65536 inclusive, indicating the minimum width of the data field. If the specified field width is too small to hold the number, the field is widened to accommodate the number. If no field width is specified, a write call uses a field just large enough to hold the value, and a read call stops at the first field delimiter it encounters.
- E**     The **E** option is only valid in read calls, where it forces early termination of a data field when a delimiter is encountered. The default delimiters are blank and comma, but other delimiters can be declared with the %"... " directive. The **E** option overrides the field width specifier.
- Z**     The **Z** option is only valid in write calls, where it means add zeros (0) to the left of the number to fill the output data field. **Z** is not valid without a field width specifier (*fw*).
- J**     The **J** option is only valid in write calls, where it means left-justify the number in the output data field. Without the **J** option, numbers are right justified in the field.
- U**     The **U** option means the number to be read or written is unsigned, and the call should ignore any evidence to the contrary. A write call will write an unsigned (and possibly very large) positive integer into the field even if the source variable appears negative. A read call will ignore plus and minus signs in the data field. The **U** option is most often used when writing via the %H and %O directives.
- S**     The **S** option means the number to be read or written has a minus sign if it is negative, and no sign if it is positive. This is the default unless the **P** option is specified. **S** is always redundant for read calls because ASCII data must be signed if negative.
- P**     The **P** option is only valid in write calls, and means prepend a minus sign to the number if it is negative, and prepend a plus sign if it is positive.
- W**     The **W** option means the number to be read or written is a 2-byte integer value. If **W** is not specified, **L** is the default.
- L**     The **L** option means the number to be read or written is a 4-byte integer value. It is not a valid option if **W** is specified.
- Y**     The **Y** option means write hexadecimal integers with lowercase letters instead of uppercase. It is only valid with hexadecimal output fields in write call control strings.

### Control String Directives

The control string directives modify a call's interpretation of the control string passed to it. The following table summarizes the control string directives:

Control String-Related Format Directives	
Directive	Function
%"..."	declare characters to be used as field delimiters
%%\$	end control string
%. .	end control string, inserting new line character
%%n(	repeat string n times start
%)	repeat string n times stop

**%"..."** The %"... " directive defines the field delimiters that read calls use when the E option is specified in a data directive. When the %"... " directive appears in a control string, the characters between the double quotes, following the percent sign, become the field delimiters for that string. If no %"... " directive appears, then the read call uses the default delimiters, comma (,) and space ( ), equivalent to %", ". For example, %", ";" adds a double quote (") and semicolon (;) to the default delimiters.

**%%\$** The %%\$ directive marks the end of a control string. It has no other effect.

**%. .** The %. . directive marks the end of a control string too. When used in a write call, it also means add a record terminator to the output. The record terminator is usually an ASCII newline character. When used in a read call, "%. ." is identical to "%. \$".

**%%n( and %) )**

Together, the %%n( and %) ) directives delimit a portion of a control string to be repeated *n* times. *N* is required, and must be an integer between 1 and 65536 inclusive. For example, the fragment "%. 5( ... %) )" is identical to writing the delimited portion, denoted here as "... " five times in succession in the control string that contains it. Repeat directives cannot be nested.

### Format Directives

The format directives modify the format of input or output between data fields. The following table summarizes the format directives:

Format-Related Format Directives	
Directive	Function
<code>%c%</code>	write a single <code>%</code>
<code>%/</code>	write a record terminator
<code>%nT</code>	tab to column <i>n</i>
<code>%nX</code>	skip <i>n</i> characters
<code>%n@</code>	move argument pointer to argument <i>n</i>

- `%c%` When a write call encounters a “`% %`” in a control string, it writes a single percent sign (`%`) to the output.
- `%/` When a write call encounters a “`%/`” in a control string, it writes a record terminator to the output. The record terminator is usually an ASCII newline character.
- `%nT` The `%nT` directive causes a read or write call to tab to column *n* before interpreting the control string further. *N* is optional; if it is omitted (`%T`), the call uses the next variable in the calling sequence for the tab value. Write calls use space as the tab fill character.
- `%nX` The `%nX` directive causes a read or write call to skip *n* spaces before interpreting the control string further. The directive causes a write call to write *n* spaces to the output, and causes a read call to skip *n* characters in the input. *N* is optional; if it is omitted (`%X`), the call uses the next variable in the calling sequence for the number of spaces.
- `%n@` The `%n@` directive causes a write call to move its argument pointer to the *n*th source argument. It allows programs interpret source arguments in an order specified by the control string, but does not generate any output. For example, “`%1@`” resets the argument pointer. *N* is required.

#### Data Types

##### `vfmt_$string_t`

A 200-byte array for passing control strings to `vfmt_$` calls.

#### Errors

##### `vfmt_$unterminated_ctl_string`

Unterminated control string.

##### `vfmt_$invalid_ctl_string`

Invalid control string.

##### `vfmt_$too_few_args`

Too few arguments supplied for a read or decode call.

##### `vfmt_$fw_required`

Field width missing on “`()`” designator.

- vfmt\_\$eos**  
Encountered end of string where more text was expected.
- vfmt\_\$null\_token**  
Encountered null token where numeric token was expected.
- vfmt\_\$nonnumeric\_char**  
Non-numeric character found where numeric was expected.
- vfmt\_\$sign\_not\_allowed**  
Sign encountered in unsigned field.
- vfmt\_\$value\_too\_large**  
Value out of range in text string.
- vfmt\_\$nonmatching\_char**  
Character in text string does not match control string.
- vfmt\_\$nonmatching\_delimiter**  
Terminator in text string does not match specified terminator.

## NAME

`vfmt_$decode10` – formatted read from a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>

int vfmt_$decode10(
    char *control_string,
    char *source_string,
    int &source_length,
    int *decode_count,
    status_$t *status,
    void *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9, *a10)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';

function vfmt_$decode10(
    in control_string: univ vfmt_$string_t;
    in source_string: univ vfmt_$string_t;
    in source_length: integer;
    out decode_count: integer;
    out status: status_$t;
    var a1, a2, a3, a4, a5, a6, a7, a8, a9, a10:
        univ vfmt_$generic_unsigned_arg): integer;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'

integer*2 nchar
parameter (nchar = 128)

integer*4 status
integer*2 return_value, source_length, decode_count
character control_string*(nchar), source_string*200
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10

last = vfmt_$decode10(control_string, source_string, source_length,
&                      decode_count, status,
&                      a1, a2, a3, a4, a5,
&                      a6, a7, a8, a9, a10)
```

## DESCRIPTION

`Vfmt_$decode10` decodes an ASCII *source\_string*, formatted into fields as defined by

*control\_string*, into 10 destination variables, and returns the number of bytes from *source\_string* decoded into the destination variables.

*control\_string*

A VFMT format string defining the fields in *source\_string* to be decoded.

*source\_string*

An ASCII string to decode.

*source\_length*

The number of bytes in *source\_string*.

*decode\_count*

The number of fields in *source\_string* decoded into the destination variables.

*status*

The completion status.

*a1, a2, a3, a4, a5, a6, a7, a8, a9, a10*

Destination variables to receive the decoded data.

#### NOTES

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$decode10` and the maximum string length specified in the call.

#### SEE ALSO

`vfmt_$decode2`, `vfmt_$decode5`.

## NAME

`vfmt_$decode2` – formatted read from a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
int vfmt_$decode2(
    char *control_string,
    char *source_string,
    int &source_length,
    int *decode_count,
    status_$t *status,
    void *a1, *a2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
function vfmt_$decode2(
    in control_string: univ vfmt_$string_t;
    in source_string: univ vfmt_$string_t;
    in source_length: integer;
    out decode_count: integer;
    out status: status_$t;
    var a1, a2: univ vfmt_$generic_unsigned_arg): integer;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 return_value, source_length, decode_count
character control_string*(nchar), source_string*200
integer*4 a1, a2
```

```
last = vfmt_$decode2(control_string, source_string, source_length,
& decode_count, status, a1, a2)
```

## DESCRIPTION

`Vfmt $decode2` decodes an ASCII *source\_string*, formatted into fields as defined by *control\_string*, into two destination variables, and returns the number of bytes from *source\_string* decoded into the destination variables.

*control\_string*

A VFMT format string defining the fields in *source\_string* to be decoded.

*source\_string*

An ASCII string to decode.

*source\_length*

The number of bytes in *source\_string*.

*decode\_count*

The number of fields in *source\_string* decoded into the destination variables.

*status*

The completion status.

*a1, a2*

Destination variables to receive the decoded data.

**NOTES**

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$decode2` and the maximum string length specified in the call.

**SEE ALSO**

`vfmt_$decode10`, `vfmt_$decode5`.

## NAME

`vfmt_$decode5` – formatted read from a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
int vfmt_$decode(
    char *control_string,
    char *source_string,
    int &source_length,
    int *decode_count,
    status_$t *status,
    void *a1, *a2, *a3, *a4, *a5)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
function vfmt_$decode5(
    in control_string: univ vfmt_$string_t;
    in source_string: univ vfmt_$string_t;
    in source_length: integer;
    out decode_count: integer;
    out status: status_$t;
    var a1, a2, a3, a4, a5:
        univ vfmt_$generic_unsigned_arg): integer;
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 return_value, source_length, decode_count
character control_string*(nchar), source_string*200
integer*4 a1, a2, a3, a4, a5
```

```
last = vfmt_$decode5(control_string, source_string, source_length,
& decode_count, status,
& a1, a2, a3, a4, a5)
```

## DESCRIPTION

`Vfmt_$decode5` decodes an ASCII `source_string`, formatted into fields as defined by `control_string`, into five destination variables, and returns the number of bytes from

*source\_string* decoded into the destination variables.

*control\_string*

A VFMT format string defining the fields in *source\_string* to be decoded.

*source\_string*

An ASCII string to decode.

*source\_length*

The number of bytes in *source\_string*.

*decode\_count*

The number of fields in *source\_string* decoded into the destination variables.

*status*

The completion status.

*a1, a2, a3, a4, a5*

Destination variables to receive the decoded data.

#### NOTES

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$decode5` and the maximum string length specified in the call.

#### SEE ALSO

`vfmt_$decode10`, `vfmt_$decode2`.

## NAME

`vfmt_$encode` – formatted write to a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>

void vfmt_$encode(
    char *control_string,
    char *string_buffer,
    int &buffer_size,
    int *string_length,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';

procedure vfmt_$encode(
    in control_string: univ vfmt_$string_t ;
    out string_buffer: univ vfmt_$string_t;
    in buffer_size: integer;
    out string_length: integer;
    in a1, a2, a3, a4, a5,
        a6, a7, a8, a9, a10,
        a11, a12, a13, a14, a15,
        a16, a17, a18, a19, a20: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'

integer*2 nchar
parameter (nchar = 128)

integer*2 buffer_size, string_length
character control_string*(nchar), string_buffer*200
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11
integer*4 a12, a13, a14, a15, a16, a17, a18, a19, a20

call vfmt_$encode(control_string, string_buffer, buffer_size, string_length,
& a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11,
& a12, a13, a14, a15, a16, a17, a18, a19, a20)
```

## DESCRIPTION

`Vfmt_$encode` encodes up to twenty source variables into an ASCII string whose format is defined by `control_string`, and writes the result into `string_buffer`.

*control\_string*

A VFMT format string defining how to format the source variables.

*string\_buffer*

A buffer allocated to receive the formatted string supplied by `vfmt_$encode`.

*buffer\_size*

The number of bytes allocated at *string\_buffer* to receive the formatted string. `Vfmt_$encode` will not write more than *buffer\_size* bytes into *string\_buffer*.

*string\_length*

The number of bytes that `vfmt_$encode` wrote into *string\_buffer*.

*a1, a2, a3, ... a19, a20*

Up to 20 source variables containing data to encode into ASCII.

**NOTES**

A string field in *control\_string* requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

**SEE ALSO**

`vfmt_$encode10`, `vfmt_$encode2`, `vfmt_$encode5`.

## NAME

`vfmt_$encode10` – formatted write to a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$encode10(
    char *control_string,
    char *string_buffer,
    int &buffer_size,
    int *string_length,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$encode10(
    in control_string: univ vfmt_$string_t ;
    out string_buffer: univ vfmt_$string_t;
    in buffer_size: integer;
    out string_length: integer;
    in a1, a2, a3, a4, a5,
       a6, a7, a8, a9, a10: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 buffer_size, string_length
character control_string*(nchar), string_buffer*200
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
```

```
call vfmt_$encode10(control_string, string_buffer,
&                    buffer_size, string_length,
&                    a1, a2, a3, a4, a5, a6, a7, a8, a9, a10)
```

## DESCRIPTION

`Vfmt_$encode10` encodes up to ten source variables into an ASCII string whose format is defined by `control_string`, and writes the result into `string_buffer`.

`control_string`

— A VFMT format string defining how to format the source variables.

*string\_buffer*

A buffer allocated to receive the formatted string supplied by `vfmt_$encode10`.

*buffer\_size*

The number of bytes allocated at *string\_buffer* to receive the formatted string. `Vfmt_$encode10` will not write more than *buffer\_size* bytes into *string\_buffer*.

*string\_length*

The number of bytes that `vfmt_$encode10` wrote into *string\_buffer*.

*a1, a2, a3, ... a9, a10*

Up to 10 source variables containing data to encode into ASCII.

**NOTES**

A string field in *control\_string* requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

**SEE ALSO**

`vfmt_$encode`, `vfmt_$encode2`, `vfmt_$encode5`.

## NAME

`vfmt_$encode2` – formatted write to a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$encode2(
    char *control_string,
    char *string_buffer,
    int &buffer_size,
    int *string_length,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$encode2(
    in control_string: univ vfmt_$string_t ;
    out string_buffer: univ vfmt_$string_t;
    in buffer_size: integer;
    out string_length: integer;
    in a1, a2: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 buffer_size, string_length
character control_string*(nchar), string_buffer*200
integer*4 a1, a2
```

```
call vfmt_$encode2(control_string, string_buffer,
&                   buffer_size, string_length,
&                   a1, a2)
```

## DESCRIPTION

`Vfmt_$encode2` encodes up to two source variables into an ASCII string whose format is defined by `control_string`, and writes the result into `string_buffer`.

`control_string`

A VFMT format string defining how to format the source variables.

*string\_buffer*

A buffer allocated to receive the formatted string supplied by `vfmt_$encode2`.

*buffer\_size*

The number of bytes allocated at *string\_buffer* to receive the formatted string. `Vfmt_$encode2` will not write more than *buffer\_size* bytes into *string\_buffer*.

*string\_length*

The number of bytes that `vfmt_$encode2` wrote into *string\_buffer*.

*a1, a2* Up to two source variables containing data to encode into ASCII.

#### NOTES

A string field in *control\_string* requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

#### SEE ALSO

`vfmt_$encode`, `vfmt_$encode10`, `vfmt_$encode5`.

## NAME

`vfmt_$encode5` – formatted write to a string

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$encode5(
    char *control_string,
    char *string_buffer,
    int &buffer_size,
    int *string_length,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$encode5(
    in control_string: univ vfmt_$string_t ;
    out string_buffer: univ vfmt_$string_t;
    in buffer_size: integer;
    out string_length: integer;
    in a1, a2, a3, a4, a5: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 buffer_size, string_length
character control_string*(nchar), string_buffer*200
integer*4 a1, a2, a3, a4, a5
```

```
call vfmt_$encode5(control_string, string_buffer,
&                   buffer_size, string_length,
&                   a1, a2, a3, a4, a5)
```

## DESCRIPTION

`Vfmt_$encode5` encodes up to five source variables into an ASCII string whose format is defined by `control_string`, and writes the result into `string_buffer`.

`control_string`

A VFMT format string defining how to format the source variables.

*string\_buffer*

A buffer allocated to receive the formatted string supplied by `vfmt_$encode5`.

*buffer\_size*

The number of bytes allocated at *string\_buffer* to receive the formatted string. `Vfmt_$encode5` will not write more than *buffer\_size* bytes into *string\_buffer*.

*string\_length*

The number of bytes that `vfmt_$encode5` wrote into *string\_buffer*.

*a1, a2, a3, a4, a5*

Up to five source variables containing data to encode into ASCII.

#### NOTES

A string field in *control\_string* requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

#### SEE ALSO

`vfmt_$encode`, `vfmt_$encode10`, `vfmt_$encode2`.

## NAME

`vfmt_$read10` – formatted read from standard input

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$read10(
    char *control_string,
    int *field_count,
    status_$t *status,
    void *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9, *a10)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$read10(
    in control_string: univ vfmt_$string_t;
    out field_count: integer;
    out status: status_$t;
    var a1, a2, a3, a4, a5, a6, a7, a8, a9, a10:
        univ vfmt_$generic_unsigned_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 field_count
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
```

```
call vfmt_$read10(control_string, field_count, status,
& a1, a2, a3, a4, a5,
& a6, a7, a8, a9, a10)
```

## DESCRIPTION

`Vfmt_$read10` reads an ASCII line from standard input, formatted in fields as defined by `control_string`, into 10 destination variables.

`control_string`

A VFMT format string defining the input fields to be decoded.

*field\_count*

The number of input fields decoded into the destination variables.

*status* The completion status.

*a1, a2, a3, a4, a5, a6, a7, a8, a9, a10*

Destination variables to receive the decoded input.

**NOTES**

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$read10` and the maximum string length specified in the call.

**SEE ALSO**

`vfmt_$read2`, `vfmt_$read5`.

## NAME

`vfmt_$read2` – formatted read from standard input

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$read2(
    char *control_string,
    int *field_count,
    status_$t *status,
    void *a1, *a2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$read2(
    in control_string: univ vfmt_$string_t;
    out field_count: integer;
    out status: status_$t;
    var a1, a2: univ vfmt_$generic_unsigned_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 field_count
character control_string*(nchar)
integer*4 a1, a2
```

```
call vfmt_$read2(control_string, field_count, status, a1, a2)
```

## DESCRIPTION

`Vfmt_$read2` reads an ASCII line from standard input, formatted in fields as defined by `control_string`, into two destination variables.

*control\_string*

A VFMT format string defining the input fields to be decoded.

*field\_count*

The number of input fields decoded into the destination variables.

*status*

The completion status.

*a1, a2* Destination variables to receive the decoded input.

**NOTES**

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$read2` and the maximum string length specified in the call.

**SEE ALSO**

`vfmt_$read10`, `vfmt_$read5`.

## NAME

`vfmt_$read5` – formatted read from standard input

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$read5(
    char *control_string,
    int *field_count,
    status $t *status,
    void *a1, *a2, *a3, *a4, *a5)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$read5(
    in control_string: univ vfmt_$string_t;
    out field_count: integer;
    out status: status $t;
    var a1, a2, a3, a4, a5: univ vfmt_$generic_unsigned_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 field_count
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5
```

```
call vfmt_$read5(control_string, field_count, status,
& a1, a2, a3, a4, a5)
```

## DESCRIPTION

`Vfmt_$read5` reads an ASCII line from standard input, formatted in fields as defined by `control_string`, into five destination variables.

`control_string`

A VFMT format string defining the input fields to be decoded.

`field_count`

The number of input fields decoded into the destination variables.

*status* The completion status.

*a1, a2, a3, a4, a5*

Destination variables to receive the decoded input.

#### NOTES

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$read5` and the maximum string length specified in the call.

#### SEE ALSO

`vfmt_$read10`, `vfmt_$read2`.

## NAME

`vfmt_$rs10` – formatted read from a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$rs10(
    int &stream_id,
    char *control_string,
    int *field_count,
    status_$t *status,
    void *a1, *a2, *a3, *a4, *a5, *a6, *a7, *a8, *a9, *a10)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$rs10(
    in stream_id: univ integer;
    in control_string: univ vfmt_$string_t;
    out field_count: integer;
    out status: status_$t;
    var a1, a2, a3, a4, a5,
        a6, a7, a8, a9, a10: univ vfmt_$generic_unsigned_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 stream_id, field_count
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
```

```
call vfmt_$rs10(stream_id, control_string, field_count, status,
& a1, a2, a3, a4, a5, a6, a7, a8, a9, a10)
```

## DESCRIPTION

`Vfmt_$rs10` reads an ASCII record from the stream specified by `stream_id`, formatted in fields as defined by `control_string`, into 10 destination variables.

`stream_id`

The stream ID of the stream to read from.

*control\_string*

A VFMT format string defining the input fields to be decoded.

*field\_count*

The number of input fields decoded into the destination variables.

*status* The completion status.

*a1, a2, a3, a4, a5, a6, a7, a8, a9, a10*

Destination variables to receive the decoded input.

**NOTES**

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by `vfmt_$$rs10` and the maximum string length specified in the call.

**SEE ALSO**

`vfmt_$$rs2`, `vfmt_$$rs5`.

## NAME

`vfmt_$$rs2` – formatted read from a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$$rs2(
    int &stream_id,
    char *control_string,
    int *field_count,
    status_$$t *status,
    void *a1, *a2)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$$rs2(
    in stream_id: univ integer;
    in control_string: univ vfmt_$$string_t;
    out field_count: integer;
    out status: status_$$t;
    var a1, a2: univ vfmt_$$generic_unsigned_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 stream_id, field_count
character control_string*(nchar)
integer*4 a1, a2
```

```
call vfmt_$$rs2(stream_id, control_string, field_count, status,
& a1, a2)
```

## DESCRIPTION

`Vfmt_$$rs2` reads an ASCII record from the stream specified by `stream_id`, formatted in fields as defined by `control_string`, into two destination variables.

`stream_id`

The stream ID of the stream to read from.

*control\_string*

A VFMT format string defining the input fields to be decoded.

*field\_count*

The number of input fields decoded into the destination variables.

*status*

The completion status.

*a1, a2*

Destination variables to receive the decoded input.

## NOTES

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by *vfmt\_\$\$rs2* and the maximum string length specified in the call.

## SEE ALSO

*vfmt\_\$\$rs10*, *vfmt\_\$\$rs5*.

## NAME

`vfmt_$rs5` – formatted read from a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$rs5(
    int &stream_id,
    char *control_string,
    int *field_count,
    status_$t *status,
    void *a1, *a2, *a3, *a4, *a5)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$rs5(
    in stream_id: univ integer;
    in control_string: univ vfmt_$string_t;
    out field_count: integer;
    out status: status_$t;
    var a1, a2, a3, a4, a5: univ vfmt_$generic_unsigned_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*4 status
integer*2 stream_id, field_count
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5
```

```
call vfmt_$rs5(stream_id, control_string, field_count, status,
& a1, a2, a3, a4, a5)
```

## DESCRIPTION

`Vfmt_$rs5` reads an ASCII record from the stream specified by `stream_id`, formatted in fields as defined by `control_string`, into five destination variables.

`stream_id`

The stream ID of the stream to read from.

*control\_string*

A VFMT format string defining the input fields to be decoded.

*field\_count*

The number of input fields decoded into the destination variables.

*status* The completion status.

*a1, a2, a3, a4, a5*

Destination variables to receive the decoded input.

**NOTES**

A string field in *control\_string* requires two destination variables: a character array to receive the value of the string, and a 2-byte integer variable to receive the number of bytes in the decoded string.

The length of a string field can be defined in *control\_string* with the **M** option or, if the **M** option is not present, by passing a maximum string length in the destination variable intended to receive the string length. Regardless of whether there is an **M** option associated with the string field, the value returned in the destination variable is the minimum of the actual length of the string supplied by *vfmt\_\$\$rs5* and the maximum string length specified in the call.

**SEE ALSO**

*vfmt\_\$\$rs10*, *vfmt\_\$\$rs2*.

## NAME

`vfmt_$write` – formatted write to standard output

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$write(
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$write(
    in control_string: univ vfmt_$string_t;
    in a1, a2, a3, a4, a5, a6, a7, a8, a9, a10,
        a11, a12, a13, a14, a15, a16, a17, a18, a19, a20:
        univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11
integer*4 a12, a13, a14, a15, a16, a17, a18, a19, a20
```

```
call vfmt_$write10(control_string,
& a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11,
& a12, a13, a14, a15, a16, a17, a18, a19, a20)
```

## DESCRIPTION

`Vfmt_$write` encodes up to twenty source variables into an ASCII string whose format is defined by `control_string`, and writes the result to standard output.

`control_string`

A VFMT format string defining how to format the source variables.

`a1, a2, a3, ... a19, a20`

Up to 20 source variables containing data to encode into ASCII.

## NOTES

A string field in `control_string` requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in

VFMT\_\$WRITE

VFMT\_\$WRITE

the string.

**SEE ALSO**

vfmt\_\$write10, vfmt\_\$write2, vfmt\_\$write5.

## NAME

`vfmt_$write10` – formatted write to standard output

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$write10(
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$write10(
    in control_string: univ vfmt_$string_t;
    in a1, a2, a3, a4, a5, a6, a7, a8, a9, a10:
        univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
```

```
call vfmt_$write10(control_string,
& a1, a2, a3, a4, a5, a6, a7, a8, a9, a10)
```

## DESCRIPTION

`Vfmt_$write10` encodes up to ten source variables into an ASCII string whose format is defined by `control_string`, and writes the result to standard output.

`control_string`

A VFMT format string defining how to format the source variables.

`a1, a2, a3, ..., a9, a10`

Up to 10 source variables containing data to encode into ASCII.

## NOTES

A string field in `control_string` requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

VFMT\_\$WRITE10

VFMT\_\$WRITE10

**SEE ALSO**

vfmt\_\$write, vfmt\_\$write2, vfmt\_\$write5.

## NAME

`vfmt_$write2` – formatted write to standard output

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$write2(
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$write2(
    in control_string: univ vfmt_$string_t;
    in a1, a2: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
character control_string*(nchar)
integer*4 a1, a2
```

```
call vfmt_$write2(control_string, a1, a2)
```

## DESCRIPTION

`Vfmt_$write2` encodes up to two source variables into an ASCII string whose format is defined by `control_string`, and writes the result to standard output.

*control\_string*

A VFMT format string defining how to format the source variables.

*a1, a2* Up to two source variables containing data to encode into ASCII.

## NOTES

A string field in `control_string` requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

## SEE ALSO

`vfmt_$write`, `vfmt_$write10`, `vfmt_$write5`.

## NAME

`vfmt_$write5` – formatted write to standard output

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$write5(
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$write5(
    in control_string: univ vfmt_$string t;
    in a1, a2, a3, a4, a5: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5
```

```
call vfmt_$write5(control_string, a1, a2, a3, a4, a5)
```

## DESCRIPTION

`Vfmt_$write5` encodes up to five source variables into an ASCII string whose format is defined by `control_string`, and writes the result to standard output.

`control_string`

A VFMT format string defining how to format the source variables.

`a1, a2, a3, a4, a5`

Up to five source variables containing data to encode into ASCII.

## NOTES

A string field in `control_string` requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

## SEE ALSO

`vfmt_$write`, `vfmt_$write10`, `vfmt_$write2`.

## NAME

`vfmt_$ws` – formatted write to a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$ws(
    ios_$id_t &stream_id,
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$ws(
    in stream_id: ios_$id_t;
    in control_string: univ vfmt_$string_t;
    in a1, a2, a3, a4, a5, a6, a7, a8, a9, a10,
        a11, a12, a13, a14, a15, a16, a17, a18, a19, a20:
        univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 stream_id
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11
integer*4 a12, a13, a14, a15, a16, a17, a18, a19, a20
```

```
call vfmt_$ws(stream_id, control_string,
&             a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11,
&             a12, a13, a14, a15, a16, a17, a18, a19, a20)
```

## DESCRIPTION

`Vfmt_$ws` encodes up to twenty source variables into an ASCII string whose format is defined by `control_string`, and writes the result to the stream specified by `stream_id`.

`stream_id`

The stream ID of the stream to write to.

`control_string`

A VFMT format string defining how to format the source variables.

*a1, a2, a3, ... a19, a20*

Up to 20 source variables containing data to encode into ASCII.

**NOTES**

A string field in *control\_string* requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

**SEE ALSO**

*vfmt\_\$ws10, vfmt\_\$ws2, vfmt\_\$ws5.*

## NAME

`vfmt_$ws10` – formatted write to a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$ws10(
    ios_sid_t &stream_id,
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$ws10(
    in stream_id: ios_sid_t;
    in control_string: univ vfmt$string_t;
    in a1, a2, a3, a4, a5, a6, a7, a8, a9, a10:
        univ vfmt$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 stream_id
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5, a6, a7, a8, a9, a10
```

```
call vfmt_$ws10(stream_id, control_string,
& a1, a2, a3, a4, a5, a6, a7, a8, a9, a10)
```

## DESCRIPTION

`Vfmt_$ws10` encodes up to ten source variables into an ASCII string whose format is defined by `control_string`, and writes the result to the stream specified by `stream_id`.

*stream\_id*

The stream ID of the stream to write to.

*control\_string*

A VFMT format string defining how to format the source variables.

*a1, a2, a3, ... a9, a10*

Up to 10 source variables containing data to encode into ASCII.

VFMT\_\$WS10

VFMT\_\$WS10

**NOTES**

A string field in *control\_string* requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

**SEE ALSO**

vfmt\_\$ws, vfmt\_\$ws2, vfmt\_\$ws5.

## NAME

`vfmt_$ws2` – formatted write to a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$ws2(
    ios_$id_t &stream_id,
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$ws2(
    in stream_id: ios_$id_t;
    in control_string: univ vfmt_$string_t;
    in a1, a2: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 stream_id
character control_string*(nchar)
integer*4 a1, a2
```

```
call vfmt_$ws2(stream_id, control_string, a1, a2)
```

## DESCRIPTION

`Vfmt_$ws2` encodes up to two source variables into an ASCII string whose format is defined by `control_string`, and writes the result to the stream specified by `stream_id`.

*stream\_id*

The stream ID of the stream to write to.

*control\_string*

A VFMT format string defining how to format the source variables.

*a1, a2*

Up to two source variables containing data to encode into ASCII.

## NOTES

A string field in `control_string` requires two source variables: a character array containing the value of the string, and a 2-byte integer variable specifying the number of bytes in

VFMT\_\$WS2

VFMT\_\$WS2

the string.

**SEE ALSO**

vfmt\_\$ws, vfmt\_\$ws10, vfmt\_\$ws5.

## NAME

`vfmt_$ws5` – formatted write to a stream

## SYNOPSIS (C)

```
#include <apollo/base.h>
#include <apollo/vfmt.h>
```

```
void vfmt_$ws5(
    ios_sid_t &stream_id,
    char *control_string,
    ...)
```

## SYNOPSIS (Pascal)

```
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/vfmt.ins.pas';
```

```
procedure vfmt_$ws5(
    in stream_id: ios_sid_t;
    in control_string: univ vfmt_$string_t;
    in a1, a2, a3, a4, a5: univ vfmt_$generic_signed_arg);
```

## SYNOPSIS (FORTRAN)

```
%include '/sys/ins/base.ins.ftn'
%include '/sys/ins/vfmt.ins.ftn'
```

```
integer*2 nchar
parameter (nchar = 128)
```

```
integer*2 stream_id
character control_string*(nchar)
integer*4 a1, a2, a3, a4, a5
```

```
call vfmt_$ws5(stream_id, control_string, a1, a2, a3, a4, a5)
```

## DESCRIPTION

`Vfmt_$ws5` encodes up to five source variables into an ASCII string whose format is defined by `control_string`, and writes the result to the stream specified by `stream_id`.

*stream\_id*  
The stream ID of the stream to write to.

*control\_string*  
A VFMT format string defining how to format the source variables.

*a1, a2, a3, a4, a5*  
Up to five source variables containing data to encode into ASCII.

## NOTES

A string field in `control_string` requires two source variables: a character array

VFMT\_\$WS5

VFMT\_\$WS5

containing the value of the string, and a 2-byte integer variable specifying the number of bytes in the string.

**SEE ALSO**

vfmt\_\$ws, vfmt\_\$ws10, vfmt\_\$ws2.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that every entry, no matter how small, should be recorded to ensure the integrity of the financial data. This includes not only sales and purchases but also expenses and income. The document provides a detailed list of items that should be tracked, such as inventory levels, customer orders, and supplier invoices. It also outlines the procedures for recording these transactions, including the use of specific forms and the assignment of responsibilities to different staff members.

The second part of the document focuses on the analysis of the recorded data. It describes various methods for identifying trends and anomalies in the financial records. This includes comparing current performance with historical data and industry benchmarks. The document also discusses the importance of regular audits and reconciliations to detect and correct any errors or discrepancies. It provides a step-by-step guide for conducting these audits, from the selection of samples to the final reporting of findings.

The final part of the document addresses the communication of financial information to management and other stakeholders. It highlights the need for clear, concise, and timely reporting. The document provides a template for financial statements and a list of key performance indicators (KPIs) that should be included in these reports. It also discusses the importance of providing context and explanations for any significant changes or fluctuations in the data. The document concludes with a summary of the key points and a call to action for all staff members to adhere to the established procedures and maintain the highest standards of accuracy and transparency.

## Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Domain/OS Call Reference, Volume 2*  
Order No.: 012888-A00  
Date of Publication: June, 1988

What type of user are you?

- |  |   |
|--|---|
| <input type="checkbox"/> System programmer; language _____       |   |
| <input type="checkbox"/> Applications programmer; language _____ |   |
| <input type="checkbox"/> System maintenance person               | <input type="checkbox"/> Manager/Professional   |
| <input type="checkbox"/> System Administrator                    | <input type="checkbox"/> Technical Professional |
| <input type="checkbox"/> Student Programmer                      | <input type="checkbox"/> Novice                 |
| <input type="checkbox"/> Other                                   |   |

How often do you use the Domain system? \_\_\_\_\_

What parts of the manual are especially useful for the job you are doing? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

What additional information would you like the manual to include? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.) \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Your Name \_\_\_\_\_ Date \_\_\_\_\_

\_\_\_\_\_  
Organization \_\_\_\_\_

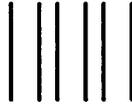
\_\_\_\_\_  
Street Address \_\_\_\_\_

\_\_\_\_\_  
City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

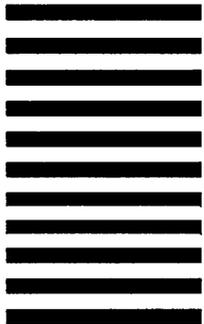
No postage necessary if mailed in the U.S.

cut or fold along dotted line

fold



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES



**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824  
POSTAGE WILL BE PAID BY ADDRESSEE

**APOLLO COMPUTER INC.**  
**Technical Publications**  
**P.O. Box 451**  
**Chelmsford, MA 01824**

fold

## Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Domain/OS Call Reference, Volume 2*

Order No.: 012888-A00

Date of Publication: June, 1988

What type of user are you?

- |                          |   |   |
|--------------------------|---|---|
| <input type="checkbox"/> | System programmer; language _____       |   |
| <input type="checkbox"/> | Applications programmer; language _____ |   |
| <input type="checkbox"/> | System maintenance person               | <input type="checkbox"/> Manager/Professional   |
| <input type="checkbox"/> | System Administrator                    | <input type="checkbox"/> Technical Professional |
| <input type="checkbox"/> | Student Programmer                      | <input type="checkbox"/> Novice                 |
| <input type="checkbox"/> | Other                                   |   |

How often do you use the Domain system? \_\_\_\_\_

What parts of the manual are especially useful for the job you are doing? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

What additional information would you like the manual to include? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.) \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Your Name Date

\_\_\_\_\_  
Organization

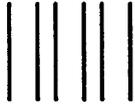
\_\_\_\_\_  
Street Address

\_\_\_\_\_  
City State Zip

No postage necessary if mailed in the U.S.

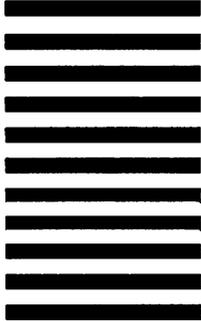
cut or fold along dotted line

fold



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824  
POSTAGE WILL BE PAID BY ADDRESSEE



**APOLLO COMPUTER INC.**  
**Technical Publications**  
**P.O. Box 451**  
**Chelmsford, MA 01824**

fold

## Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Domain/OS Call Reference, Volume 2*

Order No.: 012888-A00

Date of Publication: June, 1988

What type of user are you?

- |  |   |
|--|---|
| <input type="checkbox"/> System programmer; language _____       |   |
| <input type="checkbox"/> Applications programmer; language _____ |   |
| <input type="checkbox"/> System maintenance person               | <input type="checkbox"/> Manager/Professional   |
| <input type="checkbox"/> System Administrator                    | <input type="checkbox"/> Technical Professional |
| <input type="checkbox"/> Student Programmer                      | <input type="checkbox"/> Novice                 |
| <input type="checkbox"/> Other                                   |   |

How often do you use the Domain system? \_\_\_\_\_

What parts of the manual are especially useful for the job you are doing? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

What additional information would you like the manual to include? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.) \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Your Name \_\_\_\_\_ Date \_\_\_\_\_

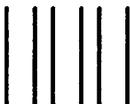
\_\_\_\_\_  
Organization \_\_\_\_\_

\_\_\_\_\_  
Street Address \_\_\_\_\_

\_\_\_\_\_  
City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

No postage necessary if mailed in the U.S.

fold



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824  
POSTAGE WILL BE PAID BY ADDRESSEE



**APOLLO COMPUTER INC.**  
Technical Publications  
P.O. Box 451  
Chelmsford, MA 01824

fold

## Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Domain/OS Call Reference, Volume 2*

Order No.: 012888-A00

Date of Publication: June, 1988

What type of user are you?

- |  |   |
|--|---|
| <input type="checkbox"/> System programmer; language _____       |   |
| <input type="checkbox"/> Applications programmer; language _____ |   |
| <input type="checkbox"/> System maintenance person               | <input type="checkbox"/> Manager/Professional   |
| <input type="checkbox"/> System Administrator                    | <input type="checkbox"/> Technical Professional |
| <input type="checkbox"/> Student Programmer                      | <input type="checkbox"/> Novice                 |
| <input type="checkbox"/> Other                                   |   |

How often do you use the Domain system? \_\_\_\_\_

What parts of the manual are especially useful for the job you are doing? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

What additional information would you like the manual to include? \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.) \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Your Name \_\_\_\_\_ Date \_\_\_\_\_

\_\_\_\_\_  
Organization \_\_\_\_\_

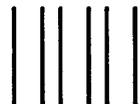
\_\_\_\_\_  
Street Address \_\_\_\_\_

\_\_\_\_\_  
City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

No postage necessary if mailed in the U.S.

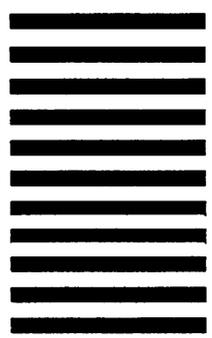
cut or fold along dotted line

fold



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824  
POSTAGE WILL BE PAID BY ADDRESSEE



**APOLLO COMPUTER INC.**  
Technical Publications  
P.O. Box 451  
Chelmsford, MA 01824

fold