

PMFX Primer

Peter T. Schwenn, Velocity

Larry Leibman, Proteus Engineering

George S. Hazen, Proteus Engineering

April 19, 1996

PMFX/PERL API	3
INTRODUCTION.....	3
THE PMFX RELATIONSHIP BETWEEN FASTSHIP AND PERL.....	3
BASICS OF PERL IN THE PMFX CONTEXT.....	3
<i>Cautions for C Programmers and Others About PMFX Syntax.....</i>	<i>5</i>
BASICS OF THE FASTSHIP / FASTYACHT SIDE OF PMFX.....	6
USER INTERACTION WITH PMFX.....	7
THE PERL / FASTSHIP GLUE: MAGIC VARIABLES	8
CAVEATS AND NOTES	14
FUTURE	14
OTHER MEANS OF EXTENDING FASTSHIP	15
<i>UnPerl Macros.....</i>	<i>15</i>
<i>Recording Macros</i>	<i>15</i>
<i>Customizing the Menu and Fast-Buttons.....</i>	<i>15</i>
<i>Compiled Routines for Permanent Incorporation.....</i>	<i>15</i>
ADVANCED PMFX USE OF FASTSHIP	17

PMFX/Perl API

Introduction

PMFX is a computer language for customizing and extending FastShip and FastYacht.¹ It is based on the C-like language Perl written by Larry Wall.² It is a tool to be used by both the users and the developers of these two marine CAD/CAM software suites. Perl is interpretive but very fast. It is familiar because of its C roots. It is non-punitive in error conditions. Its core is easy to learn, but it is very broad in its functions -- adding a good deal of networking and operating system functionality to C, along with a rich set of string manipulation functions. It is very broad compared to Shell languages in terms of offering a broad variety of types including floats and associative arrays as well as strings and integers, and also in terms of having none of the usual arbitrary and constraining limits on program and data size and structure found in equally easy-to-use languages.

This introduction is written for C programmers. After reading this, read either the tutorial "Learning Perl" by Randal Schwartz or the very comprehensive "Programming Perl" by Larry Wall and Randal L. Schwartz. Both are Nutshell Handbooks from O'Reilly & Associates, Inc. -- available anywhere technical or computer books are sold.

Note: "FastShip" will be used here to refer to any of the FXPerl contexts: FastYacht, FastShip, or V++ . FastShip documents usually refer to PMFX as Parametric FastShip. PMFX is a more general name for the marriage of Perl with FastShip, V++ or FastYacht.

Acknowledgment:

We would like to thank Larry Wall for creating and extending Perl, and for making it freely available.

The PMFX Relationship Between FastShip and Perl

PMFX is Perl extended to include virtually all of FastShip's commands except those very few which can only be performed by a live human-computer interaction.

Basics of Perl in the PMFX Context

Perl is like C, but requires no compilation by the user, and does not crash hard. It has the programming control structure of C but lacks the data Structures of C. It goes beyond C in expressing everything (and more) that a command-line shell such as DOS or UNIX's Bourne, C or Korn shells do: dealing with regular expressions, directories, with communications, with networking, with databases, with signals and with errors.

It handles types in a more automatic way than C: it has scalars (which can be treated as either strings, flags, integers or floating-point numbers.) The scalar's names always begin with "\$". It has arrays of any such

¹FastShip is a product of Proteus Engineering. FastYacht Velocity Prediction , "V++" is a product of Velocity.

²Descriptions of Perl and where to get Perl itself and other Perl materials can best be found in Programming Perl by Larry Wall and Randall Schwarz, published by O'Reilly & Associates,, 1990.

scalars (but not multi-dimensional arrays without extra work). Their names begin with "@". And it has associative arrays (which C does not have) with permit elements to be looked up by any scalar not just an integer. Their names begin with %. Perl switches gracefully between scalar and array contexts usually without troubling the user about it. User-defined function's names are defined as in **sub x** and called as in **&x(...)**. Perl offers most of the usual C-Library functions (and quite a few others.) It runs on all of the platforms that FastShip does.

It has the usual operators +, -, *, / and so on, used often in the usual sort of **\$a=<expression>**; assignment statement. It has the usual **if(){}, do, while** and **until** block control structures. But the condition for these is very often a file-reading form **<stream>** that C lacks. Perl has some additional control structure, e.g. **for each**. It has built-in string equality and inequality tests **eq** and **ne**. Going well beyond C it has all of the regular expression handling of Unix shells (but much extended and entirely regular), and virtually all of the operating system functions sometimes in the Shells.

What does a little Perl program look like? Here's one for the "flavor" -- its FXPerl's own converter subroutine which renders FXPerl palatable to Perl:

```
sub fs2Perl
#####
# Routine to convert a PMFS macro file into a full fledged Perl macro.
#####
{
  package pmfs;

  local($file_in, $file_out);
  local(*FILE, @lines);
  local($lbl, $cmd, $arg, $del, $com);
  local($indent, $space, $label, $comment);
  $file_in = shift(@_);
  $file_out = shift(@_);

  open (FILE,"<".$file_in);

  # One line at a time pattern space
  $* = 0;

  @lines=<FILE>;

  # Identify lines that contain 'native' FastShip commands. When found,
  # pass the command and all of its arguments to FastShip via the function
  # &fxcall(). Check for possible Perl labels in front of these commands and
  # retain them if present.

  $lbl = '\s*[a-z]\w*:\s*';
  $bkl = '\s*{\s*';
  $bkr = '}';
  $fxc = '&main\fscale\s*\(\s*';
  $cmd = '\b[a-z]\w*-\w+(\w\.)*\b';
  $cmd .= '\bcancel\b|\becho\b|\bexec\b|\bexit\b|\bhelp\b|\bmessage\b';
  $cmd .= '\bmove\b|\brotate\b|\bscale\b|\brecover\b|\breplay\b';
  $cmd .= '\bselect\b|\bssystem\b|\bundo\b|\bvpp\b|\bprompt\b';
  $arg = '\b[^\n!]*[w+\.!]?[^\n!;\s]*';
  $del = '\s*;\s+!\s+(-?[0-9.]+)(\s*."*)?';
  $com = '\#\s$';

  foreach (@lines)
  {
    # Remove any comments prior to further processing.

    $comment = "";
    if ( /($com)?\n/ )
    {
      $comment = $1;
      s/($com)?\n//;
    }

    # Remove any label from beginning of line.
```

```

$label = "";
if ( /($lb|)/ )
{
    $label = $1;
    s/($lb|)/;
}

# Convert any 'native' FS command lines into &fxcall subroutine call.

if (/^(($bkl)?($fxc)?($cmd)($arg)?($del)?[^\s]*($bkr)?/i)
{
    if ($2)
        {last;}

    # Expand 'native' FS's delay/prompt syntax; ie. ! xx "prompt"

    $indent = index ($_, $3);
    $space = "";
    for ($i = 0; $i < $indent; $i++) {$space = $space." ";}

    $_ = $1;
    if ($5)
    {
        if ($7)
        {
            $_ .= "&main\fscall('message $6, $7');\n";
            $_ .= $space."&main\fscall(\".$3.$4.\");$8";
        }
    }
    else
    {
        $_ .= "&main\fscall(\".$3.$4.\");$8";
    }
}

# Append any labels and comments earlier removed.

$_ = $label.$_.$comment."\n";
}

close (FILE);

# Write converted macro to $file_out.

open (FILE,">".$file_out);

foreach (@lines)
{ $last = $_; print FILE; }

# Ensure macros don't return errors; add final expression " 1;"

if ( $last ne "1;\n" )
{
    print FILE "1;\n";
}

close FILE;
}
# end fs2Perl ();

```

Cautions for C Programmers and Others About PMFX Syntax

1. There are no {}-less if's, for's etc. E.g. C's `if (a>b) max=a;` must be `if (a>b) { max=a; } .`

2. Types and Scoping are quite different from C's, learn them from scratch -- they aren't a variation or an analogy to C's.
3. Unlike C you can have single-character operators for string tests. But they aren't != and ==, they are eq and ne.
4. Yes, that ugly \$ really does go on the front of every scalar variable name. It makes for automatic typing and greater freedom of extension for built-in names for Perl. Same with function calls: they need their leading "&".
5. C programmers who forget that each statement ends with a ";" will probably do the same in Perl..

Basics of the FastShip / FastYacht side of PMFX

You just freely mix FastShip commands into Perl. There are a few extra commands created just for PMFX which don't exist in either Perl or in FastShip. PMFX takes care of converting the FastShip commands you mix in so that they are processable by Perl.

Study the following macro which cuts as many stations along the LWL as the user wants. It would be invoked as all macros can be: **read-macro <macro-name><CR>** from the FastShip command-line:

```
#####
# This macro is designed to create ?? evenly spaced stations along the
# current waterline. Delete any previously defined stations before
# creating the new definitions. Additional stations are added with
# equal spacing in the fore and aft overhangs.
#
# Note that the section definition are sensitive to the currently set
# units.
#
# Author: George Hazen
# Last modified: 9 January 1994
#####

# Delete any stations that might be defined.

delete-sections sta;

# Compute hydrostatics so as to determine the waterline endings.

measure-volume mirror /top {0};

# Set up associative arrays for current units, system and hydro values.

&get_units;
&get_sys;
&get_hydro;

# Copy geometry extents data into simple arrays.

@lwl_min = split(/,/,$hydro{'lwl_min'});
@lwl_max = split(/,/,$hydro{'lwl_max'});
@box_min = split(/,/,$hydro{'box_min'});
@box_max = split(/,/,$hydro{'box_max'});

# Convert extents data into currently set units using %units.

$utype = 'length';
$range_min = $lwl_min[0] / $units{$utype};
$range_max = $lwl_max[0] / $units{$utype};
$loa_max = $box_max[0] / $units{$utype};
$loa_min = $box_min[0] / $units{$utype};
```

```

# Prompt user for number of stations.

$NSTA = &input("How many stations do you wish along the LWL?");

# Define the $NSTA sections along the load waterline.

define-sections sta $NSTA {$RANGE_MIN, $RANGE_MAX};

# Now add stations in the ends beyond the waterline

$STA_SPC = ($RANGE_MAX - $RANGE_MIN) / ($NSTA - 1);

# Count number of stations required in forward overhang, and define them..

$SISTA = 1;
for ($XSTA = $RANGE_MAX; $XSTA < $LOA_MAX; $XSTA += $STA_SPC)
  { $SISTA++; }
if ($SISTA)
  { define-sections sta $SISTA {$RANGE_MAX, $XSTA}; }

# Count number of stations required in aft overhang, and define them..

$SISTA = 1;
for ($XSTA = $RANGE_MIN; $XSTA > $LOA_MIN; $XSTA -= $STA_SPC)
  { $SISTA++; }
if ($SISTA)
  { define-sections sta $SISTA {$XSTA, $RANGE_MIN}; }

# Finally, make sure that sections are on.

set-sections on;

# Force recover file and clearing of sectional area curve.

update-recover;

```

User Interaction with PMFX

To get data from the user, use `&input(<prompt-string>)`; . It returns as its value whatever the user types in.

To report data to the user, put it in a file and then call `FG_OutputPopup(<filename>)`. Your results will popup in a text dialog box, which because it is scrollable, will accommodate all sizes. This Popup can be printed (in full) to any operating system defined printer (or file) by hitting its Print button.

Use `&fprintf(<message>)` to print a one-liner to the FastShip message line.

In future, PMFX will provide for simple user-defined Popups for input as well.

The Perl / FastShip Glue: Magic Variables

Summary of Internal Data Available to Users of Parametric FastShip

One of the benefits offered to the user by Parametric FastShip is access to much of the data stored internally in the program. This data may be broadly categorized into three general areas: 1.) system state data, 2.) hydrostatics data, and 3.) units data. Details of the information available to the user is provided below.

System state data consists of information defining the current status of the program. Most of this information is initialized at program startup and may be changed by the user either through the FastShip graphical interface or by entering FastShip commands at the command line. Some of the data applies to the program globally, such as the viewport layout, while other data applies to the current surface model, such as the project name and description. The system state data is stored in a scalar variable called `$pmfs_sys` in the form of a somewhat long and unwieldy string. However, users are encouraged to use the utility subroutine call `&get_sys` to parse this information into an associative array called `%sys`. The keys and values stored in this associative array are summarized below in Table 1.

Hydrostatics data consists of information defining the hydrostatics properties of the current surface model. Most of the information presented in the FastShip hydrostatics output is available. Additional data not presented in the hydrostatics output is also available. All dimensional values are provided in FastShip's internal units system, the SI system. Hydrostatics are computed when the FastShip measure-volume command is issued. Hence the data available to the user applies to the state of the surface model when the most recent call to the measure-volume command was made. If only a portion of the current surface model was included in the hydrostatics calculations, then the hydrostatics data applies only to those surfaces. Also, if a list of flotation planes was provided, the last flotation plane is the one to which the data applies. The hydrostatics data is stored in a scalar variable called `$pmfs_hydro`. Users are encouraged to use the utility subroutine call `&get_hydro` to parse this information into an associative array called `%hydro`. The keys and values stored in this associative array are summarized below in Table 2.

Finally, units data consists of conversion factors and units labels for the current program state. FastShip uses SI units internally. Conversion factors represent values which when multiplied by the current user units give equivalent SI units. The user can modify any of the unit conversions through the FastShip set-units command. The units data is stored in a scalar variable called `$pmfs_units`. Users are encouraged to use utility subroutine call `&get_units` to parse this information into two associative arrays. The first array, called `%units`, contains the conversion factors for each of the units types, and the second array, called `%unit_labels`, contains the unit labels for each of the units types. The keys and values stored in these two associative arrays are summarized in Tables 3 and 4.

A good example of the use of these variables and their use can be seen in this excerpt from the `lwlsta` macro shown earlier:

```
# Set up associative arrays for current units, system and hydro values.

&get_units;
&get_sys;
&get_hydro;

# Copy geometry extents data into simple arrays.

@lwl_min = split(/,/,$hydro{'lwl_min'});
@lwl_max = split(/,/,$hydro{'lwl_max'});
@box_min = split(/,/,$hydro{'box_min'});
@box_max = split(/,/,$hydro{'box_max'});

# Convert extents data into currently set units using %units.

$utype = 'length';
$range_min = $lwl_min[0] / $units{$utype};
$range_max = $lwl_max[0] / $units{$utype};
$loa_max = $box_max[0] / $units{$utype};
```

```
$loa_min = $box_min[0] / $units{$utype};
```

The functions `&get_units`, `&get_sys`, and `&get_hydro` are first used to parse the information into the associative arrays `%units`, `%sys`, and `%hydro` respectively. Then scalar arrays for `lwl` and `box` min and max dimensions (X, Y, and Z values) are taken from the `hydro` array, and finally the length value from `units` array is used to convert various values to the current unit system.

Table 1: Contents of the %sys Associative Array

Key	Description of Contents
recover_freq	recover frequency; defines the frequency with which the surface model is automatically backed up in terms of the number of destructive commands between saves; 0 means autosaving is turned off
sys_id	current application name
customer	customer identification string used in printout of hydrostatics, etc.
prog_path	current prog path; same as config_path at the present time
config_path	current config path
data_path	current data path
filename	filename (root only) for the current surface model
project	project name
description	project description
current_part	current edit part
modified	modified flag indicating whether or not the current surface model has been modified from its original state when read in (1 = modified, 0 = unmodified)
active	value in the range from 0 to 3 defining the active viewport
layout	value in the range from 0 to 3 defining the viewport layout
abs_rel	absolute/relative flag defining editing mode (1 = absolute, 0 = relative)
sign	system sign (1 = positive, -1 = negative)
xyz_cyl	xyz/cylindrical flag defining coordinate mode (1 = xyz, 0 = cylindrical)
zup	positive z-direction (1 = up, -1 = down)
io_user_units	I/O user units flag (1 = user-defined units on file output, 0 = internal system units on file output)
primary_edit	primary editing constraint plane (0 = user-defined, 1 = station, 2 = buttock, 3 = waterline)
secondary_edit	secondary editing constraint plane (0 = user-defined, 1 = station, 2 = buttock, 3 = waterline)
lls_hpy	view control mode flag (1 = latitude/longitude/spin, 0 = heel/pitch/yaw)
selection_buffer	selection buffer flag (currently not used in FastShip 5)
surface_quill	unused flag
surface_prop	surface curvature calculation flag (1 = calculate surface curvature, 0 = don't calculate surface curvature)
surface_curve_type	surface curvature type (0 = none, 1 = minimum curvature, 2 = maximum curvature, 3 = gaussian curvature, 4 = average curvature)
false_color	false color flag (1 = use false coloring when rendering surface curvature, 0 = don't use false color)
anchor	comma-delimited list containing x, y, and z values of the current system anchor for relative editing operations
home	comma-delimited list containing x, y, and z values of the current system home for editing operations
handle	comma-delimited list containing x, y, and z values of the current system handle for editing operations
origin	comma-delimited list containing x, y, and z values of the current system origin for editing operations
mesh_div	system default mesh division setting used for creating new surfaces
mesh_wts	comma-delimited list containing major and minor mesh line weights
multi_net_wt	line weight in screen pixels used for rendering of multiple net line segments
quill_norm	scale or normalization factor which is applied to curvature quills during rendering of line or surface curvature; a value of 1 indicates that the magnitude of curvature is equal to the length of the quill measured in current units system
hsr_algorithm	integer flag defining how solid rendering is done; 0 indicates software z-buffering is off while 1 indicates software z-buffering is used for solid rendering

Table 2: Contents of the %hydro Associative Array

Key	Description of Contents
ax	immersed section area at the station of maximum area
bm_long	longitudinal metacentric radius using mesh-based integration of volume
bm_tran	transverse metacentric radius using mesh-based integration of volume
box_max	comma-delimited list containing maximum x, y, and z values of the box bounding all surfaces included in the most recent hydrostatics calculation
box_min	comma-delimited list containing minimum x, y, and z values of the box bounding all surfaces included in the most recent hydrostatics calculation
cb	comma-delimited list containing x, y, and z values of the center of buoyancy using mesh-based integration of volume
c_block	block coefficient using mesh-based integration of volume
c_pris	longitudinal prismatic coefficient using mesh-based integration of volume
c_pris_aft	longitudinal prismatic coefficient for surfaces aft of the station of maximum area using mesh-based integration of volume
c_pris_fwd	longitudinal prismatic coefficient for surfaces forward of the station of maximum area using mesh-based integration of volume
c_wp	waterplane coefficient for the current floatation plane
c_x	maximum section coefficient
gm_long	longitudinal metacentric height using mesh-based integration of volume (only valid when center of gravity has been specified)
gm_tran	transverse metacentric height using mesh-based integration of volume (only valid when center of gravity has been specified)
lwl_max	comma-delimited list containing maximum x, y, and z values of the rectangle bounding flotation planes for all surfaces included in the most recent hydrostatics calculation
lwl_min	comma-delimited list containing minimum x, y, and z values of the rectangle bounding flotation planes for all surfaces included in the most recent hydrostatics calculation
mc_long	height of the longitudinal metacenter relative to the flotation plane using mesh-based integration of volume
mc_tran	height of the transverse metacenter relative to the flotation plane using mesh-based integration of volume
moment	comma-delimited list containing first x, y, and z moments of volume about the origin using mesh-based integration of volume
sa_cb	comma-delimited list containing x, y, and z values of the center of buoyancy using section area-based integration of volume
sa_cb_aft	comma-delimited list containing x, y, and z values of the center of buoyancy of volume aft of the station of max area using section area-based integration of volume
sa_cb_fwd	comma-delimited list containing x, y, and z values of the center of buoyancy of volume forward of the station of max area using section area-based integration of volume
sa_inertia	comma-delimited list containing x, y, and z moments of inertia of volume about the origin using section-area based integration of volume
sa_inertia_aft	comma-delimited list containing x, y, and z moments of inertia of volume aft of the station of max area about the origin using section-area based integration of volume
sa_inertia_fwd	comma-delimited list containing x, y, and z moments of inertia of volume forward of the station of max area about the origin using section-area based integration of volume
sa_moment	comma-delimited list containing first x, y, and z moments of volume about the origin using section-area based integration of volume
sa_moment_aft	comma-delimited list containing first x, y, and z moments of volume aft of the station of max area about the origin using section-area based integration of volume
sa_moment_fwd	comma-delimited list containing first x, y, and z moments of volume forward of the station of max area about the origin using section-area based integration of volume
sa_volume	submerged volume based on integration of the section area curve
sa_volume_aft	submerged volume aft of the station of max area based on integration of the section area curve
sa_volume_fwd	submerged volume forward of the station of max area based on integration of the section area curve
sa_wetsurf	wetted surface area based on longitudinal integration of section girths
sa_xinertia	comma-delimited list containing third x, y, and z moments of volume about the origin

Key	Description of Contents
	using section-area based integration of volume
sa_xinertia_aft	comma-delimited list containing third x, y, and z moments of volume aft of the station of max area about the origin using section-area based integration of volume
sa_xinertia_fwd	comma-delimited list containing third x, y, and z moments of volume forward of the station of max area about the origin using section-area based integration of volume
volume	submerged volume based on integration of the surface mesh
wet_max	comma-delimited list containing maximum x, y, and z values of the box bounding the wetted portions of all surfaces included in the most recent hydrostatics calculations
wet_min	comma-delimited list containing minimum x, y, and z values of the box bounding the wetted portions of all surfaces included in the most recent hydrostatics calculations
wetsurf	wetted surface area based on integration of surface mesh panels
wp_area	waterplane area for the current flotation plane
wp_ixx	moment of inertia of the waterplane about the z axis for the current flotation plane
wp_mom	first moment of the waterplane about the z axis for the current flotation plane
xax	x location of the station of maximum area

Table 3: Contents of the %units Associative Array

Key	Description of Contents
none	1.0
length	conversion factor to convert current length units to SI units
area	conversion factor to convert current area units to SI units
volume	conversion factor to convert current volume units to SI units
inertia	conversion factor to convert current inertia units to SI units
mass	conversion factor to convert current mass units to SI units
force	conversion factor to convert current force units to SI units
time	conversion factor to convert current time units to SI units
velocity	conversion factor to convert current velocity units to SI units
acceleration	conversion factor to convert current acceleration units to SI units
weight	conversion factor to convert current weight units to SI units
density	conversion factor to convert current density units to SI units

Table 4: Contents of %unit_labels Associative Array

Key	Description of Contents
none	empty string ("")
length	length units label (e.g. "m")
area	area units label (e.g. "m^2")
volume	volume units label (e.g. "m^3")
inertia	inertia units label (e.g. "m^4")
mass	mass units label (e.g. "kg")
force	force units label (e.g. "N")
time	time units label (e.g. "s")
velocity	velocity units label (e.g. "m/s")
acceleration	acceleration units label (e.g. "m/s^2")
weight	weight units label (e.g. "kgf")
density	density units label (e.g. "kgf/m^3")

Caveats and Notes

1. A PMFX program has a compilation stage which first transforms it into pure Perl and then compiles the Perl for the Perl interpreter. The PMFX program of type “.mac” has as its pure Perl form a new file called “.pmc”. If both a “.mac” and a “.pmc” exist for a given Perl program, then the “.pmc” will be run, unless the “.mac” has a later time stamp. In this case, a new “.pmc” will be created and run.

If the program is also a data file such as when a V++'s master file “*.mf” is converted to “*.mfc”, then if V++ has written out a new “*.mf” it will have also deleted the preceding “*.mfc”. But if there has been a problem in producing the new “*.mf” the user must decide whether to delete the *.mfc or move it to *.mf as the new original.

As a final example, if new software is distributed, any new .mac's will be used instead of older .pmc's. Keep this in mind, and store your old .mac's and .pmc's elsewhere before updating your FastShip software if you want to keep them intact.

Complex as this seems it will seldom be a problem -- just keep in mind whether you need to use the source program or the translated one, and if the source, delete the translation (*.pmc or *.mfc) if the system has not automatically done so.

Future

1. PMFX is currently based on Perl 4 and FastShip 5. In future it will move on to Perl 5 which is a compatible extension to Perl 4 in very much the same OOP sense that C++ is an extension of C. Sharing variables across the Perl/C barrier will be easier and more general (for example, arrays and Structures will be shareable.)

2. PMFX does not currently offer a user controlled GUI on a par with that native to FastShip 5. As much as possible this will be realized in future -- but it must be remembered that the full power of the FS5 GUI is obtained at the cost of a very complex software development process that cannot be opened to the user entirely (and he would not want it to be in any case.)

Other Means of Extending FastShip

UnPerl Macros

Macros do not need to contain Perl expressions or be written in Perl format -- they may be simply a list of FastShip commands (which include the necessary prompt and delay commands to make such a list useful.) Needless to say pure FS macros lack the power of the control structures - and much else - of PMFX.

Recording Macros

Such FS macros can be recorded from keyboard actions. Look under the Utility pull down menu to start.

Customizing the Menu and Fast-Buttons

If you'd like to have a menu button activate any command (existing or one you've written and added) or any macro (whether it be a simple macro or an FXPerl program), examine *fastuser.def* in the config subdirectory and replace any of the existing button definitions with the one you'd like. The file has all of the models you'll need for the correct format. *Note that this can only be done in the Unix version.*

In both the Unix and Windows version, the FastUser buttons (the 14 buttons in the lower left corner of the screen) may be modified to activate any FastShip command or macro. In addition, a new set of buttons may be loaded at any time. In this case, one button is usually dedicated to loading an alternate set of buttons. The default set of buttons is defined in the config subdirectory in the *fastuser.def* file. Alternate sets can be loaded with the command **redefine-fastuser** *filename*. The format is very simple, a quick look at *fastuser.def* will make it clear.

Compiled Routines for Permanent Incorporation

This can be done on a contract basis with either Proteus Engineering or with Velocity. It might be best done for some module which needs to be faster than Perl can be. It is potentially more error prone and certainly more demanding in terms of subsidiary tools: compilers, etc.

In summary that would proceed at various levels as follow:

How to add a Magic Variable

Take xRESULT as an example. Its central definition is in data.h. To add a magic variable you must make a similar entry in data.h. Now examine ex3.mus. You will see how xRESULT appears in three positions there: one to get an enum index, one to effect its being made known as a special case in the Perl symbol table, and one to make its type known and perform the PERL->C conversion required to make it available on the Perl side (i.e. the FXPerl macro side.) If it is also to be modified on the C side when it has been modified on the Perl side, it appears in a fourth position in ex3.mus.

How to add a command

Given that you have written the command's C code, add its name to both fs4.h and command.h. They are included here. The models in them are simple and sufficient fully describe how to add new entries. Note that (currently) the entries in those files are in alphabetic order.

There are several simple prototypes for commands in command.c -- examine any subroutine with the arguments "(selection, arglist)" which form is a requirement for any new command which is to be user executable.

How to add a subsystem

A new subsystem which takes FastShip as a submodule is simply a set of new C user commands as described above which calls on FastShip routines and may be called by new macros and which may be built partly on new macros as controlling as opposed to services routines.

These new commands must be built into FastShip (The special case of Parametric FastShip or "FXPerl") through a three component make. The core may be called fslib and constitutes the core routines of FastShip. The second are the C code for the new commands. These may be directly included in the building of the lib fslib or may be built into a separate library as desired for compilation efficiency.

The last combining phase is a make which uses the Perl library routine "mis" to build the FXPerl glue routine ex3.c from the source ex3.mus. And subsequently to combine ex3.o[bj] with both the Perl objects and the above mentioned libs and the HOOPS libs.

You may be building on the PC under NT or DOS or on Unix. The makefiles are very similar but do have differences. Models for makefiles on the PC for DOS are included here as mkfslib and mkfsPerl which can be initiated via mkfsPerl.bat which is also included.

Contact Velocity or Proteus if you need Unix makefile prototypes.

You will find it very helpful to study the structures of fs4.h and their initialization in main.h, as well as command.c and the included fragment of main.c to understand the overall structure and operation of FastShip.

Advanced PMFX Use of FastShip

Study **fgen_lib.pl** for complex and near complete exploitation of FXPerl for modifying the shape of a shape of a vessel in desired ways while controlling the desired hydrostatic parameters: Cp, Bmax,