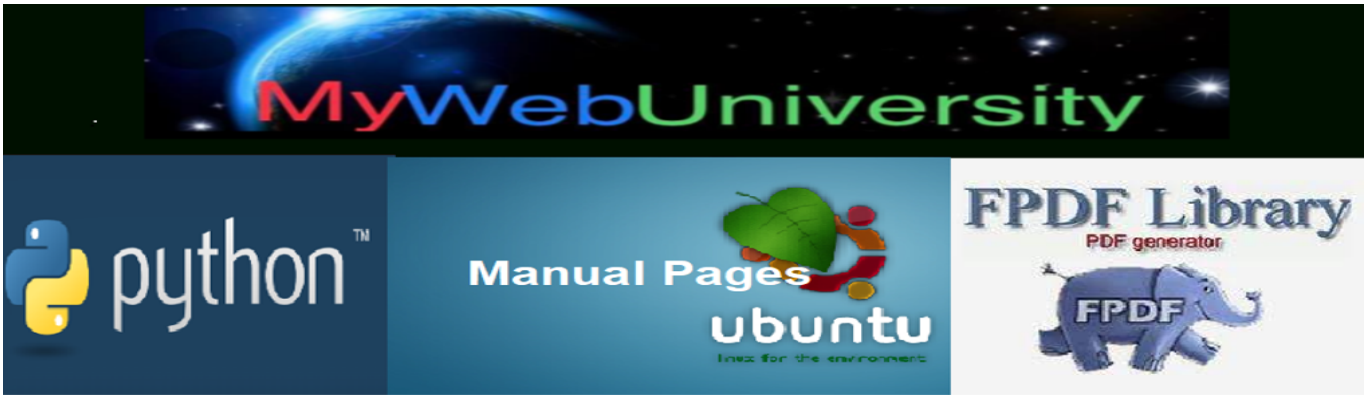

Rocky Enterprise Linux 9.2 Manual Pages on command 'Glib::GenPod.3pm'

$ man Glib::GenPod.3pm

Glib::GenPod(3pm) User Contributed Perl Documentation Glib::GenPod(3pm)

NAME

Glib::GenPod - POD generation utilities for Glib-based modules

SYNOPSIS

```
use Glib::GenPod;

# use the defaults:
xsdoc2pod ($xsdocparse_output_file, $destination_dir);

# or take matters into your own hands
require $xsdocparse_output_file;
foreach my $package (sort keys %$data) {
   print "=head1 NAME\n\n$package\n\n";
   print "=head1 METHODS\n\n" . podify_methods ($package) . "\n\n";
}
```

DESCRIPTION

This module includes several utilities for creating pod for xs-based Perl modules which build on the Glib module's foundations. The most important bits are the logic to convert the data structures created by xsdocparse.pl to describe xsubs and pods into method docs, with call signatures and argument descriptions, and converting C type names into Perl type

names. The rest of the module is mostly boiler-plate code to format and pretty-print information that may be queried from the Glib type system.

To make life easy for module maintainers, we also include a do-it-all function, xsdoc2pod(), which does pretty much everything for you. All of the pieces it uses are publically usable, so you can do whatever you like if you don't like the default output.

DOCUMENTING THE XS FILES

All of the information used as input to the methods included here comes from the XS files of your project, and is extracted by Glib::ParseXSDoc's "xsdocparse". This function creates an file containing Perl code that may be eval'd or require'd to recreate the parsed data structures, which are a list of pods from the verbatim C portion of the XS file (the xs api docs), and a hash of the remaining data, keyed by package name, and including the pods and xsubs read from the rest of each XS file following the first MODULE line.

Several custom POD directives are recognized in the XSubs section. Note that each one is sought as a paragraph starter, and must follow a "=cut" directive.

=for object Package::Name

All xsubs and pod from here until the next object directive or MODULE line will be placed under the key 'Package::Name' in xsdocparse's data structure. Everything from this line to the next "=cut" is included as a description POD.

=for object Package::Name (Other::Package::Name)

Generate POD in Package::Name but for the package Other::Package::Name. This is useful if you want POD to appear in a different namespace but still want the automatically generated hierarchy, signal and property listing, etc. from the original namespace. For example:

```
=for object Gnome2::PanelApplet::main (Gnome2::PanelApplet)
=cut
```

This will create Gnome2/PanelApplet/main.pod containing the automatically generated documentation for Gnome2::PanelApplet (hierarchy, signals, etc.) plus the method listing from the current XS file.

=for enum Package::Name

=for flags Package::Name

This causes xsdoc2pod to call "podify_values" on Package::Name when writing the pod for the current package (as set by an object directive or MODULE line). Any text in this paragraph, to the next "=cut", is included in that section.

=for deprecated_by Package::Name

Used to add a deprecation warning, indicating Package::Name as an alternative way to achieve the same functionality. There may be any number these in each package.

=for see_also some_thing_to_see

Used to add extra see alsos onto the end of the parents, if any, for a given object. Anything following the space behind see_also up to the end of the line will be placed onto the list of "see also"s. There may be any number of these in each package.

=for apidoc

=for apidoc Full::Symbol::name

Paragraphs of this type document xsubs, and are associated with the xsubs by xsdocparse.pl. If the full symbol name is not included, the paragraph must be attached to the xsub declaration (no blank lines between "=cut" and the xsub).

Within the apidoc PODs, we recognize a few special directives (the "for\s+" is optional on these):

=for signature ...

Override the generated call signature with the ... text. If you include multiple signature directives, they will all be used. This is handy when you want to change the return type or list different ways to invoke an overloaded method, like this:

```
    =for apidoc

    =signature bool Class->foo

    =signature ($thing, @other) = $object->foo ($it, $something)

    Text in here is included in the generated documentation.
    You can actually include signature and arg directives
    at any point in this pod -- they are stripped after.
    In fact, any pod is valid in here, until the =cut.

    =cut
    void foo (...)
       PPCODE:
         /* crazy code follows */
```

```
=for arg name (type) description
=for arg name description
```

The arg directive adds or overrides an argument description. The description text is optional, as is the type specification (the part in parentheses). If you want to hide an argument, specify "__hide__" as its type. The arg name does not need to include a sigil, as dollar signs will be added. FIXME what about @ for lists?

Also, we honor a couple of "modifiers" on the =for apidoc line, following the symbol name, if present:

- __hide__

  Do not document this xsub. This is handy in certain situations, e.g., for private functions. DESTROY always has this turned on, for example.

- __gerror__

  This function or method can generate a Glib::Error exception.

- __function__

  Generate a function-style signature for this xsub. The default is to generate method-style signatures.

- __deprecated__

  This function or method is deprecated and should not be used in newly written code.

(These are actually handled by Glib::ParseXSDoc, but we list them here because, well, they're an important part of how you document the XS files.)

FUNCTIONS

xsdoc2pod ($datafile, $outdir='blib/lib', index=undef)

Given a $datafile containing the output of xsdocparse.pl, create in $outdir a pod file for each package, containing everything we can think of for that module. Output is controlled by the "=for object" directives and such in the source code.

If you don't want each package to create a separate pod file, then use this function's code as a starting point for your own pretty-printer.

add_types (@filenames)

Parse the given @filenames for entries to add to the %basic_types used for C type name to Perl package name mappings of types that are not registered with the Glib type system. The file format is dead simple: blank lines are ignored; /#.*$/ is stripped from each line as comments; the first token on each line is considered to be a C type name, and the remaining tokens are the description of that type. For example, a valid file may look like this:

```
# a couple of special types
FooBar      Foo::Bar
Frob        localized frobnicator
```

C type decorations such as "const" and "*" are implied (do not include them), and the _ornull variant is handled for you.

$string = podify_properties ($packagename)

Pretty-print the object properties owned by the Glib::Object derivative $packagename and return the text as a string. Returns undef if there are no properties or $package is not a Glib::Object.

$string = podify_child_properties ($packagename)

Pretty-print the child properties owned by the Gtk2::Container derivative $packagename and return the text as a string. Returns undef if there are no child properties or $package is not a Gtk2::Container or similar class with a "list_child_properties()" method.

$string = podify_style_properties ($packagename)

Pretty-print the style properties owned by the Gtk2::Widget derivative $packagename and return the text as a string. Returns undef if there are no style properties or $package is not a Gtk2::Widget or similar class with a "list_style_properties()" method.

$string = podify_values ($packagename)

List and pretty-print the values of the GEnum or GFlags type $packagename, and return the text as a string. Returns undef if $packagename isn't an enum or flags type.

$string = podify_signals ($packagename)

Query, list, and pretty-print the signals associated with $packagename. Returns the text as a string, or undef if there are no signals or $packagename is not a Glib::Object derivative.

$string = podify_deprecated_by ($packagename, @deprecated_by)

Creates a deprecation warning for $packagename, suggesting using the items inside @deprecated_by instead.

$string = podify_pods ($pods, $position)

Helper function to allow specific placement of generic pod within the auto generated pages. Pod sections starting out with =for position XXX, where XXX is one of the following will be placed at a specified position. In the case of pod that is to be placed after a particular section that doesn't exist, that pod will be still be placed there.

This function is called at all of the specified points through out the process of generating pod for a page. Any pod matching the position passed will be returned, undef if no matches were found. If position is undef all pods without specific position information will be returned. pods is a reference to an array of pod hashes.

? SYNOPSIS

After the NAME section

? DESCRIPTION

After the SYNOPSIS section.

? post_hierarchy

After the HIERARCHY section.

? post_interfaces

After the INTERFACE section.

? post_methods

After the METHODS section.

? post_properties

After the PROPERTIES section.

? post_signals

After the SIGNALS section.

? post_enums

After the ENUMS AND FLAGS section.

? SEE_ALSO

Replacing the autogenerated SEE ALSO section completely.

? COPYRIGHT

Replacing the autogenerated COPYRIGHT section completely.

$string = podify_ancestors ($packagename)

Pretty-prints the ancestry of $packagename from the Glib type system's point of view. This uses Glib::Type->list_ancestors; see that function's docs for an explanation of why that's different from looking at @ISA.

Returns the new text as a string, or undef if $packagename is not a registered GType.

$string = podify_interfaces ($packagename)

Pretty-print the list of GInterfaces that $packagename implements. Returns the text as a string, or undef if the type implements no interfaces.

$string = podify_methods ($packagename)

Call "xsub_to_pod" on all the xsubs under the key $packagename in the data extracted by xsdocparse.pl.

Returns the new text as a string, or undef if there are no xsubs in $packagename.

$string = podify_see_alsos (@entries)

Creates a list of links to be placed in the SEE ALSO section of the page. Returns undef if nothing is in the input list.

$string = get_copyright

Returns a string that will/should be placed on each page. You can control the text of this string by calling the class method set_copyright.

If no text has been set, we will attempt to create one for you, using what has been passed to set_year, set_authors, and set_main_mod. The year defaults to the current year, the authors default to 'The Gtk2-Perl Team', and the main mod is empty by default. You want the main mod to be set to the main module of your extension for the SEE ALSO section, and on the assumption that a decent license notice can be found in that module's doc, we point the reader there.

So, in general, you will want to specify at least one of these, so that you don't credit your work to us under the LGPL.

To set them do something similar to the following in the first part of your postamble section in Makefile.PL. All occurrences of
 in the copyright are replaced with newlines, to make it easier to put in a multi-line string.

```
POD_SET=Glib::GenPod::set_copyright(qq{Copyright 1999 team-foobar<br>LGPL});
```

Glib::MakeHelper::postamble_docs_full() does this sort of thing for you.

Helpers

$perl_type = convert_type ($ctypestring)

Convert a C type name to a Perl type name.

Uses %Glib::GenPod::basic_types to look for some known basic types, and uses Glib::Type->package_from_cname to look up the registered package corresponding to a C type name. If no suitable mapping can be found, this just returns the input string.

$string = xsub_to_pod ($xsub, $sigprefix='')

Convert an xsub hash into a string of pod describing it. Includes the call signature, argument listing, and description, honoring special switches in the description pod (arg and signature overrides).

$string = compile_signature ($xsub)

Given an xsub hash, return a string with the call signature for that xsub.

$string = fixup_arg_name ($name)

Prepend a $ to anything that's not the literal ellipsis string '...'.

fixup_default

Mangle default parameter values from C to Perl values. Mostly, this does NULL => undef.

convert_arg_type

C type to Perl type conversion for argument types.

convert_return_type_to_name

C type to Perl type conversion suitable for return types.

SEE ALSO

Glib::ParseXSDoc

AUTHORS

muppet bashed out the xsub signature generation in a few hours on a wednesday night when band practice was cancelled at the last minute; he and ross mcfarland hacked this module together via irc and email over the next few days.

COPYRIGHT AND LICENSE