NAME

Glib::version − Library Versioning Utilities

SYNOPSIS

```
# require at least version 1.021 of the Glib module
use Glib '1.021';

# g_set_application_name() was introduced in GLib 2.2.0, and
# first supported by version 1.040 of the Glib Perl module.
if ($Glib::VERSION >= 1.040 and Glib->CHECK_VERSION (2,2,0)) {
   Glib::set_application_name ('My Cool Program');
}
```

DESCRIPTION

Both the Glib module and the GLib C library are works-in-progress, and their interfaces grow over time. As more features are added to each, and your code uses those new features, you will introduce version-specific dependencies, and naturally, you'll want to be able to code around them. Enter the versioning API.

For simple Perl modules, a single version number is sufficient; however, Glib is a binding to another software library, and this introduces some complexity. We have three versions that fully specify the API available to you.

Perl Bindings Version

Perl modules use a version number, and Glib is no exception. *$Glib::VERSION* is the version of the current Glib module. By ad hoc convention, gtk2−perl modules generally use version numbers in the form x.yyz, where even yy values denote stable releases and z is a patchlevel.

```
$Glib::VERSION
use Glib 1.040; # require at least version 1.040
```

Compile-time ("Bound") Library Version

This is the version of the GLib C library that was available when the Perl module was compiled and installed. These version constants are equivalent to the version macros provided in the GLib C headers. GLib uses a major.minor.micro convention, where even minor versions are stable. (gtk2−perl does not officially support unstable versions.)

```
Glib::MAJOR_VERSION
Glib::MINOR_VERSION
Glib::MICRO_VERSION
Glib->CHECK_VERSION($maj,$min,$mic)
```

Run-time ("Linked") Library Version

This is the version of the GLib C library that is available at run time; it may be newer than the compile-time version, but should never be older. These are equivalent to the version variables exported by the GLib C library.

```
Glib::major_version
Glib::minor_version
Glib::micro_version
```

Which one do I use when?

Where do you use which version? It depends entirely on what you're doing. Let's explain by example:

o Use the Perl module version for bindings support issues

You need to register a new enum for use as the type of an object property. This is something you can do with all versions of the underlying C library, but which wasn't supported in the Glib Perl module until `$Glib::VERSION` >= 1.040.

o Use the bound version for library features

You want to call Glib::set_application_name to set a human-readable name for your application (which is used by various parts of Gtk2 and Gnome2). **g_set_application_name()** (the underlying C function) was added in version 2.2.0 of glib, and support for it was introduced into the Glib Perl module in Glib version 1.040. However, you can build the Perl module against any stable 2.x.x version of glib, so you might not have that function available even if your Glib module is new enough! Thus, you need to check two things to see if the this function is available:

```
if ($Glib::VERSION >= 1.040 && Glib->CHECK_VERSION (2,2,0)) {
    # it's available, and we can call it!
    Glib::set_application_name ('My Cool Application');
}
```

Now what happens if you installed the Perl module when your system had glib 2.0.6, and you upgraded glib to 2.4.1? Wouldn't **g_set_application_name()** be available? Well, it's there, under the hood, but the bindings were compiled when it wasn't there, so you won't be able to call it! That's why we check the "bound" or compile-time version. By the way, to enable support for the new function, you'd need to reinstall (or upgrade) the Perl module.

o Use the linked version for runtime work-arounds

Suppose there's a function whose API did not change, but whose implementation had a bug in one version that was fixed in another version. To determine whether you need to apply a workaround, you would check the version that is actually being used at runtime.

```
if (Glib::major_version == 2 &&
    Glib::minor_version == 2 &&
    Glib::micro_version == 1) {
   # work around bug that exists only in glib 2.2.1.
}
```

In practice, such situations are very rare.

METHODS

boolean = Glib->CHECK_VERSION ($required_major, `$required_minor`, `$required_micro`)

- `$required_major` (integer)
- `$required_minor` (integer)
- `$required_micro` (integer)

Provides a mechanism for checking the version information that Glib was compiled against. Essentially equvilent to the macro GLIB_CHECK_VERSION.

(MAJOR, MINOR, MICRO) = Glib->GET_VERSION_INFO

Shorthand to fetch as a list the glib version for which Glib was compiled. See `Glib::MAJOR_VERSION`, etc.

integer = Glib::MAJOR_VERSION

Provides access to the version information that Glib was compiled against. Essentially equivalent to the #define's GLIB_MAJOR_VERSION.

integer = Glib::MICRO_VERSION

Provides access to the version information that Glib was compiled against. Essentially equivalent to the #define's GLIB_MICRO_VERSION.

integer = Glib::MINOR_VERSION

Provides access to the version information that Glib was compiled against. Essentially equivalent to the #define's GLIB_MINOR_VERSION.

integer = Glib::major_version

Provides access to the version information that Glib is linked against. Essentially equivalent to the global variable glib_major_version.

integer = Glib::micro_version

Provides access to the version information that Glib is linked against. Essentially equivalent to the global variable glib_micro_version.

integer = Glib::minor_version

Provides access to the version information that Glib is linked against. Essentially equivalent to the global variable glib_minor_version.

SEE ALSO

Glib

COPYRIGHT