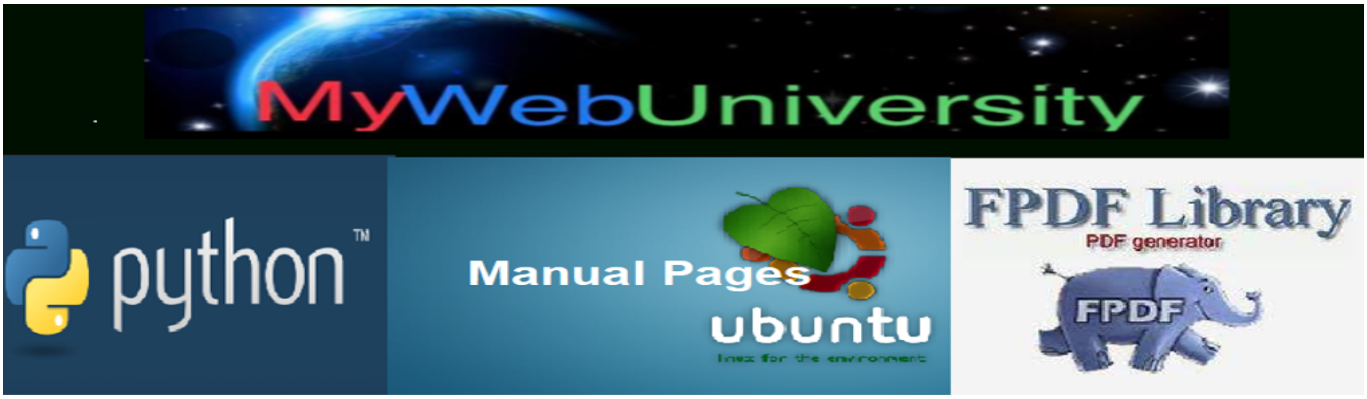

Rocky Enterprise Linux 9.2 Manual Pages on command 'DBD::Gofer::Policy::Base.3pm'

$ man DBD::Gofer::Policy::Base.3pm

DBD::Gofer::Policy::Base(3pm) User Contributed Perl Documentation DBD::Gofer::Policy::Base(3pm)

NAME

DBD::Gofer::Policy::Base - Base class for DBD::Gofer policies

SYNOPSIS

```
$dbh = DBI->connect("dbi:Gofer:transport=...;policy=...", ...)
```

DESCRIPTION

DBD::Gofer can be configured via a 'policy' mechanism that allows you to fine-tune the number of round-trips to the Gofer server. The policies are grouped into classes (which may be subclassed) and referenced by the name of the class.

The DBD::Gofer::Policy::Base class is the base class for all the policy classes and describes all the individual policy items.

The Base policy is not used directly. You should use a policy class derived from it.

POLICY CLASSES

Three policy classes are supplied with DBD::Gofer:

DBD::Gofer::Policy::pedantic is most 'transparent' but slowest because it makes more round-trips to the Gofer server.

DBD::Gofer::Policy::classic is a reasonable compromise - it's the default policy.

DBD::Gofer::Policy::rush is fastest, but may require code changes in your applications.

Generally the default "classic" policy is fine. When first testing an existing application with Gofer it is a good idea to start with the "pedantic" policy first and then switch to "classic" or a custom policy, for final testing.

POLICY ITEMS

These are temporary docs: See the source code for list of policies and their defaults.

In a future version the policies and their defaults will be defined in the pod and parsed out at load-time.

See the source code to this module for more details.

POLICY CUSTOMIZATION

XXX This area of DBD::Gofer is subject to change.

There are three ways to customize policies:

Policy classes are designed to influence the overall behaviour of DBD::Gofer with existing, unaltered programs, so they work in a reasonably optimal way without requiring code changes. You can implement new policy classes as subclasses of existing policies.

In many cases individual policy items can be overridden on a case-by-case basis within your application code. You do this by passing a corresponding "<go_<policy_name">> attribute into DBI methods by your application code. This lets you fine-tune the behaviour for special cases.

The policy items are implemented as methods. In many cases the methods are passed parameters relating to the DBD::Gofer code being executed. This means the policy can implement dynamic behaviour that varies depending on the particular circumstances, such as the particular statement being executed.

AUTHOR

Tim Bunce, <http://www.tim.bunce.name>

LICENCE AND COPYRIGHT