GNUe Common: A Developer's Introduction

A Guide to Developing Applications with GNUe Common

Version 0.4.1

Written by James Thompson

Table of Contents

Introduction

The GNUe Common is the base of almost all applications developed in the GNUe project. Like most applications in GNUe however it is designed to be usefull outside of the GNUe project.

Before designing an application using GNUe Common, the developer should be somewhat familiar with a few key concepts:

- *Python* - GNUe uses Python for almost all it's applications. There is a short section entitled "A Brief Introduction to Python" in this guide that can serve as a starting point.
- Object Oriented Programming - GNUe Common relies heavily upon OOP concepts. Without at least a rudementary grasp of these concepts programming with GNUe Common could prove somewhat difficult in the beginning.

More advanced features of GNUe Common will require some familiarity with

- *XML* - GNUe extensively uses XML for its internal storage format. While it is possible to create GNUe applications via Designer without interacting with the XML formats, a good, solid understanding of XML basics would definitely be useful.

Basic Concepts

TODO

Data Sources

TODO

Application Configuration Files

TODO

Command Line Arguments

TODO

Designing for Multiple Architectures

TODO

Hello World

No tutorial for a programming environment would be completely without a functional Hello World application. In this chapter, we will create our hello world application.

The Initial Code

Lets take a look at our complete helloworld application. Create a file named helloworld with the following contents.

```
#
# Setup the environment to know where gnue is installed
#
import sys,os
sys.path.append('/usr/local/gnue/lib/python')
os.environ['INSTALL_LIB']='/usr/local/gnue/lib/python'
os.environ['INSTALL_PREFIX']='/usr/local/gnue'

#
#Import the base application support
#
from gnue.common.GClientApp import *

#
# Define our application
#
class Hello(GClientApp):
  VERSION = "0.0.1"
  COMMAND = "helloworld"
  NAME = "Hello World"
  USAGE = GClientApp.USAGE
  SUMMARY = _("App to display the text Hello World .")
  AUTHOR = "GNU Enterprise Project"
  EMAIL = "info@gnue.org"
  REPORT_BUGS_TO = "Report bugs to info@gnue.org."

  def run(self):
    print “Hello World!”

if __name__ == '__main__':
  Hello().run()
```

After you have created your helloworld file, proceed to execute it.

```
bash-2.05a$ python helloworld
Hello World!
bash-2.05a$
```

That isn't very much output for such a large amout of code. What's going on here? Let's rerun the application using the following command.

```
bash-2.05a$ python helloworld -help
Hello World
Version 0.0.1

GNUe Common Version 0.4.1a

Usage:  helloworld [options]

App to display the text Hello World .


Available command line options:

  --configuration-options
                         Displays a list of valid configuration file entries,
                         their purpose, and their default values.

  --connections <loc>    Specifies the location of the connection definition
                         file. <loc> may specify a file name
                         (/usr/local/gnue/etc/connections.conf),or a URL
                         location (http://localhost/connections.conf).If this
                         option is not specified, the environent variable
                         GNUE_CONNECTIONS is checked.

  --debug-file <file>    Sends all debugging messages to a specified file
                         (e.g., "--debug-file trace.log" sends all output to
                         "trace.log")

  --debug-level <level>
                         Enables debugging messages. Argument specifies the
                         level of messages to display (e.g., "--debug-level 5"
                         displays all debugging messages at level 5 or below.)

  --generate-man-page    Generates a groff-formatted man page as a file in the
                         current directory.

  --help                 Displays this help screen.

  --interactive-debugger
                         Run the app inside Python's built-in debugger

  --profile              Run Python's built-in profiler and display the
                         resulting run statistics.

  --version              Displays the version information for this program.

Please report any bugs to info@gnue.org.
bash-2.05a$
```

Whoa. Do all those options really work? You bet they do. Our helloworld application has already supports an integrated debugger, profiler, and a fair bit of self documention. Now we will look at the various sections of code and explain their function.

```
#
# Setup the environment to know where gnue is installed
#
import sys,os
sys.path.append('/usr/local/gnue/lib/python')
os.environ['INSTALL_LIB']='/usr/local/gnue/lib/python'
os.environ['INSTALL_PREFIX']='/usr/local/gnue'
```

These lines deal with a shortcomming in the GNUe Common library. The current release of GNUe common (0.4.1) is unable to determine where it is installed. This probrem will be addressed in a future release.

```
#
#Import the base application support
#
from gnue.common.GClientApp import *

#
# Define our application
#
class Hello(GClientApp):
```

All applications using GNUe Common are based upon one of two different classes, GClientApp or GserverApp. GClientApp should be used when your application will not be required to run as a daemon process. If you have such a requirement then you will use GserverApp. Our helloworld application is based upon GClientApp.

```
  VERSION = "0.0.1"
  COMMAND = "helloworld"
  NAME = "Hello World"
  USAGE = GClientApp.USAGE
  SUMMARY = _("App to display the text Hello World .")
  AUTHOR = "GNU Enterprise Project"
  EMAIL = "info@gnue.org"
  REPORT_BUGS_TO = "Report bugs to info@gnue.org."
```

These lines are used to set default values displayed on various help screens and documentation formats. Looking closer at the SUMMARY definition you will notice the text string is surrounded by _(). GNUe Common contains full support for i18n[1]. For now we will not be examining that support, it will be saved for a later chapter.

```
  def run(self):
    print "Hello World!"

if __name__ == '__main__':
  Hello().run()
```

The actual application code.

Command Line Option Support

Our helloworld is a success but it seem seems to be missing something. GNUe developers are scattered all over the globe, but helloworld only displays it's message english. Lets make a few changes to the code to add the following features

- The word *world* will be replaced with any name that is passed in it on the command line.
- An -w(--welcome) option will be added to toggle on the printing of the word Welcome
- An -L(--language) option will be added to allow the user to choose the language in which we display our welcome.[2] We will support engish and maori[3] as our valid languages., as a tribute to all the andrews from New Zealand, maori as our example languages.

Here is the new code with the bolded sections containing the altered code.

```
#
# Setup the environment to know where gnue is installed
#
import sys,os
sys.path.append('/usr/local/gnue/lib/python')
os.environ[ 'INSTALL_LIB']='/usr/local/gnue/lib/python'
os.environ[ 'INSTALL_PREFIX']='/usr/local/gnue'

#
#Import the base application support
#
from gnue.common.GClientApp import *

#
# Define our application
#
class Hello(GClientApp):
    VERSION = "0.0.1"
    COMMAND = "helloworld"
    NAME = "Hello World"
    COMMAND_OPTIONS = [
        ['welcome_option','w','welcome',0,0,None,
         'Display the welcome'
         ],
        ['lang_option','L','lang',1,'english','language',
         'The language to use to print welcome '+ \
```

1 Internationalization
2 True i18n support would be too complicated for our simple example.
3 A tribute to numerous Andrew's from New Zealand that seem to find their way to our IRC channel.

```
          'Valid values: english, maori'
          ]
         ]
    USAGE = GClientApp.USAGE + ' [name]'
    SUMMARY = _("App to display the text Hello World .")
    AUTHOR = "GNU Enterprise Project"
    EMAIL = "info@gnue.org"
    REPORT_BUGS_TO = "Report bugs to info@gnue.org."

    def run(self):
        greetings = { 'english' : 'Welcome',
                      'maori' : 'Kia Ora'
                      }

        if self.ARGUMENTS:
            print "Hello %s!" % self.ARGUMENTS[0]
        else:
            print "Hello World!"

        if self.OPTIONS['welcome_option']:
            print greetings[self.OPTIONS['lang_option']];

if __name__ == '__main__':
    Hello().run()
```

Let's rerun the application using the following command.

```
bash-2.05a$ python helloworld –help
```

You will notice a few more options listed in the help screen and on man pages generated via the -generate-man-page option.

```
  --lang <language>, -L
                      The language to use to print thank youValid values:
                      english, maori

 --welcome, -w        Display the welcome
```

Now lets try running helloworld with a few different options.

```
bash-2.05a$ python helloworld.py
Hello World!
bash-2.05a$ python helloworld.py -w
Hello World!
Welcome
bash-2.05a$ python helloworld.py -w -L maori
Hello World!
Kia Ora
bash-2.05a$ python helloworld.py -w -L maori andrew
Hello andrew!
Kia Ora
bash-2.05a$
```

The key to adding options is the code

```
    COMMAND_OPTIONS = [
        ['welcome_option','w','welcome',0,0,None,
         'Display the welcome'
         ],
        ['lang_option','L','lang',1,'english','language',
         'The language to use to print welcome '+ \
         'Valid values: english, maori'
         ]
        ]
```

COMMAND_OPTIONS is a list of option entries that will be added to your application's list of valid options. Each option entry is a list with the following format *[key name, short option letter, long option (--) name, require argument?, default value, argument name, description]*

List position	Description
Key name	The key name that will be avaliabe in the self.OPTION dictionary when the application is executing.
Short option letter	The single letter to assign to this option
Long option (--) name	The long optoin name. The is prepended with -- on the command line.
Require value	Does this option require a value to be assigned from the command line.
Default value	If this option does require an value then this is the default value if the option is not passed in via the command line.
Value name	This is used when generating the outputing help text. In our example above you will notice --lang <language>. The word 'language' inside the <> is set by this value.
description	The description of the option displayed in help text.

Program Debugging Output Support

It is often handy to track what is going on inside a running copy of a program. You could start helloworld using the --interactive-debugger option and be dropped directly into python's debugger. This is a powerfull way to track what is happening inside your applicatoin but it can also be inconvieniet for simple debugging tasks. It is also a bit too much to ask of an end user that is requiring technical assistance..

Let's tweak our program a bit. In the interest of saving trees we'll only show small pieces of code with the new parts in bold.

```
#
#Import the base application support
#
from gnue.common.GClientApp import *
from gnue.common  inport Gdebug
```

Here we load GNUe Common's Gdebug system.

```
if self.ARGUMENTS:
  GDebug.printMesg(5,'The value to be printed is %s' % self.ARGUMENTS[0])
  print "Hello %s!" % self.ARGUMENTS[ 0]
else:
  GDebug.printMesg(5,'The default value will be printed')
  print "Hello World!"
```

Gdebug.printMesg will output to either a screen or a file depending upon the options passed to the program at start. The first argument specifies the debug level required before the text will print. The 5 in the example means that a --debug-level of 5 or higher must be specified on the command line. The GNUe project typically uses values between 1 and 10. However there is no numerical limit to how high you can set the debug level.

Configuration File Support

It is often nice to allow application options to be set at various levels. GNUe Common supports a configuration file system with the following features

- Application default settings in the code.
- A systemwide configuration file that can be overridden by individual user configuration files.
- A systemwide configuration file that can not be overridden by individual user configuration files.
- Auto-documentation support. All GNUe Common apps support a command line option (--configuration-options) that displays all valid configuration file options and their default values.

We will now alter our helloworld application to allow users to replace the world Hello with the text of their choice. Here is a complete copy of our application with the required changes in bold.

```
#
# Setup the environment to know where gnue is installed
#
import sys,os
sys.path.append('/usr/local/gnue/lib/python')
os.environ['INSTALL_LIB']='/usr/local/gnue/lib/python'
os.environ['INSTALL_PREFIX']='/usr/local/gnue'

#
#Import the base application support
#
from gnue.common.GClientApp import *
from gnue.common import Gdebug
from gnue.common import Gtypecast

#
# Define our application
#
class Hello(GClientApp):
    VERSION = "0.0.1"
    COMMAND = "helloworld"
    NAME = "Hello World"
    COMMAND_OPTIONS = [
        ['welcome_option','w','welcome',0,0,None,
         'Display the welcome'
         ],
        ['lang_option','L','lang',1,'english','language',
         'The language to use to print thank you'+ \
         'Valid values: english, maori'
         ]
        ]
    USAGE = GClientApp.USAGE + ' [name]'
    SUMMARY = _("App to display the text Hello World .")
    AUTHOR = "GNU Enterprise Project"
    EMAIL = "info@gnue.org"
    REPORT_BUGS_TO = "Report bugs to info@gnue.org."

    def __init__(self):
        ConfigOptions = (
            { 'Name'       : 'greetingText',
              'Type'       : 'Setting',
              'Comment'    : 'Use the basic editor for triggers',
              'Description': 'Use the basic editor for triggers',
              'Typecast'   : Gtypecast.text,
              'Default'    : 'Hello' },
            )

        GClientApp.__init__(self, application="helloworld"
                            defaults=ConfigOptions)
    def run(self):
      greetings = {  'english' : 'Welcome',
                     'maori' : 'Kia Ora'
                                      }

      if self.ARGUMENTS:
          GDebug.printMesg(5,'The value to be printed is %s' %
                           self.ARGUMENTS[ 0] )
          print "%s %s!" % (gConfig('greetingText'),self.ARGUMENTS[0])
      else:
          GDebug.printMesg(5,'The default value will be printed')
          print "%s World!" % gConfig('greetingText')

      if self.OPTIONS[ 'welcome_option'] :
          print greetings[ self.OPTIONS[ 'lang_option']] ;

if __name__ == '__main__':
    Hello().run()
```

Appendix A: Trigger Hierarchy

Common supports

Appendix B: Schema Definition Elements

TODO

Schema Tags

schema

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
author	*text*		No description provided
description	*text*		No description provided
title	*text*		No description provided
version	*text*		No description provided

Child Nodes

`data`, `tables`

Data Tags

data

No description provided

Child Nodes

`tabledata`

row

No description provided

Child Nodes

`value`

rows

No description provided

Child Nodes

`row`

tabledata

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED
tablename	*text*		NO DESCRIPTION PROVIDED

Child Nodes

`rows`

value

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
field	*text*		NO DESCRIPTION PROVIDED
type	*text*	`text`	NO DESCRIPTION PROVIDED

Tables Tags

tables

NO DESCRIPTION PROVIDED

Child Nodes

`import-table`, `table`

constraint

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED
type	*text*		NO DESCRIPTION PROVIDED

Child Nodes

`constraintfield`, `constraintref`

constraintfield

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED

constraintref

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		No description provided
table	*text*		No description provided

constraints

No description provided

Child Nodes

`constraint`

field

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		No description provided
type	*text*		No description provided
auto	`Y`, `N`	`N`	No description provided
default	*text*		No description provided
defaultwith	`constant`, `serial`, `timestamp`	`constant`	No description provided
description	*text*		No description provided
length	*number*		No description provided
nullable	`Y`, `N`	`Y`	No description provided
precision	*number*	`0`	No description provided

fields

No description provided

Child Nodes

`field`, `import-field`

indexes

No description provided

Child Nodes

`index`

indexfield

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED

pkfield

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED

primarykey

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED

Child Nodes

`pkfield`

table

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		NO DESCRIPTION PROVIDED
description	*text*		NO DESCRIPTION PROVIDED

Child Nodes

`constraints, fields, import-fields, indexes, primarykey`

Import Tags

import-field

NO DESCRIPTION PROVIDED

Attributes

Attribute	***Values***	***Default***	***Description***
library	*text*		No description provided
name	*text*		No description provided
type	*text*		No description provided
auto	`Y, N`	`N`	No description provided
default	*text*		No description provided
defaultwith	`constant, serial, timestamp`	`constant`	No description provided
description	*text*		No description provided
length	*number*		No description provided
nullable	`Y, N`	`Y`	No description provided
precision	*number*	`0`	No description provided

import-fields

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
library	*text*		No description provided

import-table

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
library	*text*		No description provided
name	*text*		No description provided
description	*text*		No description provided

index

No description provided

Attributes

Attribute	***Values***	***Default***	***Description***
name	*text*		No description provided
unique	`Y, N`	`N`	No description provided

Child Nodes

`indexfield`

Appendix C: ??? Objects

Appendix D: Data Objects

Alphabetical Index