GNOME Shell 3.4 DBus Interface

Finnbarr P. Murphy

(fpm@fpmurphy.com)

Many people are unaware that the GNOME Shell has a DBus interface which can be used to programmatically interact with the GNOME Shell.

Here is the DBus interface for GNOME Shell 3.4 (see *shellDBus.js*):

```
const GnomeShellIface = <interface name="org.gnome.Shell">
<method name="Eval">
    <arg type="s" direction="in" name="script" />
    <arg type="b" direction="out" name="success" />
    <arg type="s" direction="out" name="result" />
</method>
<method name="ListExtensions">
    <arg type="a{sa{sv}}" direction="out" name="extensions" />
</method>
<method name="GetExtensionInfo">
    <arg type="s" direction="in" name="extension" />
    <arg type="a{sv}" direction="out" name="info" />
</method>
<method name="GetExtensionErrors">
    <arg type="s" direction="in" name="extension" />
    <arg type="as" direction="out" name="errors" />
</method>
<method name="ScreenshotArea">
    <arg type="i" direction="in" name="x"/>
    <arg type="i" direction="in" name="y"/>
    <arg type="i" direction="in" name="width"/>
    <arg type="i" direction="in" name="height"/>
    <arg type="b" direction="in" name="flash"/>
    <arg type="s" direction="in" name="filename"/>
    <arg type="b" direction="out" name="success"/>
</method>
<method name="ScreenshotWindow">
    <arg type="b" direction="in" name="include_frame"/>
    <arg type="b" direction="in" name="include_cursor"/>
    <arg type="b" direction="in" name="flash"/>
    <arg type="s" direction="in" name="filename"/>
    <arg type="b" direction="out" name="success"/>
</method>
<method name="Screenshot">
    <arg type="b" direction="in" name="include_cursor"/>
    <arg type="b" direction="in" name="flash"/>
    <arg type="s" direction="in" name="filename"/>
    <arg type="b" direction="out" name="success"/>
</method>
<method name="FlashArea">
    <arg type="i" direction="in" name="x"/>
    <arg type="i" direction="in" name="y"/>
    <arg type="i" direction="in" name="width"/>
    <arg type="i" direction="in" name="height"/>
</method>
<method name="EnableExtension">
    <arg type="s" direction="in" name="uuid"/>
</method>
<method name="DisableExtension">
    <arg type="s" direction="in" name="uuid"/>
</method>
<method name="InstallRemoteExtension">
```



```
    <arg type="s" direction="in" name="uuid"/>
    <arg type="s" direction="in" name="version"/>
</method>
<method name="UninstallExtension">
    <arg type="s" direction="in" name="uuid"/>
    <arg type="b" direction="out" name="success"/>
</method>
<method name="LaunchExtensionPrefs">
    <arg type="s" direction="in" name="uuid"/>
</method>
<property name="OverviewActive" type="b" access="readwrite" />
<property name="ApiVersion" type="i" access="read" />
<property name="ShellVersion" type="s" access="read" />
<signal name="ExtensionStatusChanged">
    <arg type="s" name="uuid"/>
    <arg type="i" name="state"/>
    <arg type="s" name="error"/>
</signal>
</interface>;
```

As you can see, it is a mix of methods, properties and signals,

By the way, you can always use utilities like *dbus-send* or *gdbus* to retrieve the interfaces without the need to examine the source code.

```
dbus-send --session --type=method_call --print-reply  \
          --dest=org.gnome.Shell /org/gnome/Shell \
          org.freedesktop.DBus.Introspectable.Introspect
```

Python has good support for DBus. For example, here is a simple Python script which retrieves and displays the GNOME Shell version and GNOME Shell API version via the GNOME Shell DBus interface:

```
#!/usr/bin/python
try:
    import os
    import sys
    import argparse
except ImportError, detail:
    print detail
    sys.exit(1)
from gi.repository import Gio
EXTENSION_IFACE = 'org.gnome.Shell'
EXTENSION_PATH  = '/org/gnome/Shell'
class ExtensionTool:
    def __init__(self):
        try:
            self.bus = Gio.bus_get_sync(Gio.BusType.SESSION, None)
            self.proxy = Gio.DBusProxy.new_sync( self.bus, Gio.DBusProxyFlags.NONE, None,
                 EXTENSION_IFACE, EXTENSION_PATH, 'org.freedesktop.DBus.Properties', None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
    def get_shell_version(self):
        output = self.proxy.Get('(ss)', EXTENSION_IFACE, 'ShellVersion')
        return output
    def get_extension_api_version(self):
        output = self.proxy.Get('(ss)', EXTENSION_IFACE, 'ApiVersion')
        return output
def main():
    s = ExtensionTool()
    shellversion = s.get_shell_version()
    apiversion = s.get_extension_api_version()
```



```
    print 'GNOME shell version: %s' % (shellversion)
    print 'Extension API version: %s' % (apiversion)
if __name__ == "__main__":
    main()
```

If you are unfamilar with the *(ss) GVariant* notation, here is a quick introduction. A GLib *GVariant* is a general purpose data structure container, with serializing/deserializing capabilities, that stores data in a binary form. The data can be simple, like numeric values or strings, or more complex like arrays of simple types or tuples of other valid types. The data is described by a format string. This format string can be a single letter, e.g. “(s)” for a string, (b) for boolean or “(i)” for a 32 bit integer, or a longer string, e.g. “(as)” for a string array or “(asib)” for a tuple consisting of a string array, an integer and a boolean.

Here is what is outputted when the above code is executed:

```
GNOME shell version: 3.4.0
Extension API version: 1
```

You can also use utilities such as *dbus-send* to interact with GNOME Shell. For example, here is how to output information about all the installed GNOME Shell extensions.

```
$ dbus-send --session --print-reply --reply-timeout=2000    --type=method_call --dest=org.
gnome.Shell    /org/gnome/Shell    org.gnome.Shell.ListExtensions
method return sender=:1.296 -> dest=:1.388 reply_serial=2
   array [
      dict entry(
         string "user-theme@gnome-shell-extensions.gcampax.github.com"
         array [
            dict entry(
               string "uuid"
               variant                    string "user-theme@gnome-shell-extensions.gcampax.
github.com"
            )
            dict entry(
               string "extension-id"
               variant                    string "user-theme"
            )
            dict entry(
               string "settings-schema"
               variant                    string "org.gnome.shell.extensions.user-theme"
            )
            dict entry(
               string "gettext-domain"
               variant                    string "gnome-shell-extensions"
            )
            dict entry(
               string "name"
               variant                    string "User Themes"
            )
            dict entry(
               string "description"
               variant                    string "Load shell themes from user directory"
            )
            dict entry(
               string "url"
               variant                    string "http://git.gnome.org/gnome-shell-extensio
ns"
            )
            dict entry(
               string "type"
```



```
               variant                    double 1
            )
            dict entry(
               string "state"
               variant                    double 6
            )
            dict entry(
               string "path"
               variant                    string "/usr/share/gnome-shell/extensions/user-th
eme@gnome-shell-extensions.gcampax.github.com"
            )
            dict entry(
               string "error"
               variant                    string ""
            )
            dict entry(
               string "hasPrefs"
               variant                    boolean false
            )
         ]
      )
      dict entry(
         string "weather@fpmurphy.com"
         array [
            dict entry(
               string "version"
               variant                    string "2.5"
            )
            dict entry(
               string "uuid"
               variant                    string "weather@fpmurphy.com"
            )
            dict entry(
               string "name"
               variant                    string "Weather"
            )
            dict entry(
               string "description"
               variant                    string "Display current weather information and f
ive day forecast"
            )
            dict entry(
               string "url"
               variant                    string "http://fpmurphy.com/gnome-shell-extension
s"
            )
            dict entry(
               string "original-authors"
               variant                    string "Finnbarr P. Murphy"
            )
            dict entry(
               string "localedir"
               variant                    string "localedir"
            )
            dict entry(
               string "type"
               variant                    double 2
            )
            dict entry(
               string "state"
               variant                    double 6
            )
            dict entry(
               string "path"
               variant                    string "/home/fpm/.local/share/gnome-shell/extens
ions/weather@fpmurphy.com"
            )
            dict entry(
```



```
               string "error"
               variant                  string ""
            )
            dict entry(
               string "hasPrefs"
               variant                  boolean false
            )
         ]
      )
      dict entry(
         string "helloworld@example.com"
         array [
            dict entry(
               string "uuid"
               variant                  string "helloworld@example.com"
            )
            dict entry(
               string "name"
               variant                  string "Hello World"
            )
            dict entry(
               string "description"
               variant                  string "Simple example of a GNOME 3.4 extension"
            )
            dict entry(
               string "type"
               variant                  double 2
            )
            dict entry(
               string "state"
               variant                  double 6
            )
            dict entry(
               string "path"
               variant                  string "/home/fpm/.local/share/gnome-shell/extens
ions/helloworld@example.com"
            )
            dict entry(
               string "error"
               variant                  string ""
            )
            dict entry(
               string "hasPrefs"
               variant                  boolean false
            )
         ]
      )
   ]
$
```

The following Python script lists out all the installed GNOME Shell extensions together with the extension state and type.

```
#!/usr/bin/python
try:
    import os
    import sys
except ImportError, detail:
    print detail
    sys.exit(1)
from gi.repository import Gio, GLib
extstate = { 1:"enabled",
             2:"disabled",
             3:"error",
             4:"out of date",
```



```
             5:"downloading",
             6:"initialized",
            99:"uninstalled"}
exttype  = { 1:"system",
             2:"per user"}
EXTENSION_IFACE = 'org.gnome.Shell'
EXTENSION_PATH  = '/org/gnome/Shell'
class ExtensionTool:
    def __init__(self):
        try:
            self.bus = Gio.bus_get_sync(Gio.BusType.SESSION, None)
            self.proxy = Gio.DBusProxy.new_sync( self.bus, Gio.DBusProxyFlags.NONE, None,
                 EXTENSION_IFACE, EXTENSION_PATH, EXTENSION_IFACE, None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
    def list_extensions(self):
        return self.proxy.ListExtensions()
def main():
    s = ExtensionTool()
    l = s.list_extensions()
    for k, v in l.iteritems():
        print '%s: %s, %s' % (v["uuid"], extstate[v["state"]], exttype[v["type"]])
    print
if __name__ == "__main__":
    main()
```

Here is sample ouput from this example:

```
$ ./example2
user-theme@gnome-shell-extensions.gcampax.github.com: initialized, system
weather@fpmurphy.com: initialized, per user
helloworld@example.com: initialized, per user
```

Here is Python script that programmatically takes a snapshot of the current focused window in GNOME Shell:

```
#!/usr/bin/python
try:
    import os
    import sys
    import argparse
except ImportError, detail:
    print detail
    sys.exit(1)
from gi.repository import Gio, GLib
EXTENSION_IFACE = 'org.gnome.Shell'
EXTENSION_PATH  = '/org/gnome/Shell'
class ExtensionTool:
    def __init__(self):
        try:
            self.bus = Gio.bus_get_sync(Gio.BusType.SESSION, None)
            self.proxy = Gio.DBusProxy.new_sync( self.bus, Gio.DBusProxyFlags.NONE, None,
                 EXTENSION_IFACE, EXTENSION_PATH, EXTENSION_IFACE, None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
    def screenshot_window(self, frame, cursor, flash, file):
        output = self.proxy.ScreenshotWindow('(bbbs)', frame, cursor, flash, file )
        return output
def main():
    s = ExtensionTool()
    result = s.screenshot_window(True, True, True, "/tmp/screenshot.png")
if __name__ == "__main__":
```



```
    main()
```

Finally, here is the updated source code to my *extension-tool* for GNOME 3.4:

```
#!/usr/bin/python
#
#   Copyright (c) Finnbarr P. Murphy 2012.  All rights reserved.
#
#      Name: extension-tool
#
#   Version: 1.5 (05/21/2012)
#
#   License: Attribution Assurance License (see www.opensource.org/licenses)
#
import os
import sys
import argparse
from gi.repository import Gio
extstate = { 1:"enabled",
             2:"disabled",
             3:"error",
             4:"out of date",
             5:"downloading",
             6:"initialized",
            99:"uninstalled"}
exttype  = { 1:"system",
             2:"per user"}
EXTENSION_IFACE = 'org.gnome.Shell'
EXTENSION_PATH  = '/org/gnome/Shell'
ENABLED_EXTENSIONS_KEY = 'enabled-extensions'
class GnomeShell:
    def __init__(self):
        try:
            self.bus = Gio.bus_get_sync(Gio.BusType.SESSION, None)
            self.proxy1 = Gio.DBusProxy.new_sync( self.bus, Gio.DBusProxyFlags.NONE, None,
                 EXTENSION_IFACE, EXTENSION_PATH, EXTENSION_IFACE, None)
            self.proxy2 = Gio.DBusProxy.new_sync( self.bus, Gio.DBusProxyFlags.NONE, None,
                 EXTENSION_IFACE, EXTENSION_PATH, 'org.freedesktop.DBus.Properties', None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
            sys.exit(1)
    def list_extensions(self):
        return self.proxy1.ListExtensions()
    def enable_extension(self, uuid):
        return self.proxy1.EnableExtension('(s)', uuid)
    def disable_extension(self, uuid):
        return self.proxy1.DisableExtension('(s)', uuid)
    def get_shell_version(self):
        return self.proxy2.Get('(ss)', EXTENSION_IFACE, 'ShellVersion')
    def get_api_version(self):
        return self.proxy2.Get('(ss)', EXTENSION_IFACE, 'ApiVersion')
def enable_extension(uuid):
    settings = Gio.Settings(schema='org.gnome.shell')
    extensions = settings.get_strv(ENABLED_EXTENSIONS_KEY)
    if uuid in extensions:
        print >> sys.stderr, "%r is already enabled." % (uuid)
        sys.exit(1)
    s = GnomeShell()
    s.enable_extension(uuid);
    print "%r is now enabled." % (uuid)
def disable_extension(uuid):
    settings = Gio.Settings(schema='org.gnome.shell')
    extensions = settings.get_strv(ENABLED_EXTENSIONS_KEY)
    if uuid not in extensions:
        print >> sys.stderr, "%r is not enabled or installed." % (uuid,)
```

```
        sys.exit(1)
    s = GnomeShell()
    s.disable_extension(uuid);
    print "%r is now disabled." % (uuid,)
def disable_all_extensions():
    settings = Gio.Settings(schema='org.gnome.shell')
    extensions = settings.get_strv(ENABLED_EXTENSIONS_KEY)
    s = GnomeShell()
    for uuid in extensions:
        print "disabling %s" % (uuid)
        s.disable_extension(uuid);
    print "All extensions are now disabled."
def list_extensions():
    s = GnomeShell()
    l = s.list_extensions()
    for k, v in l.iteritems():
        print '%s: %s, %s' % (v["uuid"], extstate[v["state"]], exttype[v["type"]])
    print
def list_versions():
    s = GnomeShell()
    print "Extension Tool Version: %s" % "1.5"
    print "GNOME Shell Version: %s" % s.get_shell_version()
    print "GNOME Shell API Version: %s" % s.get_api_version()
def main():
    parser = argparse.ArgumentParser(description="GNOME Shell extension tool")
    group = parser.add_mutually_exclusive_group()
    group.add_argument("-d", "--disable", nargs=1, action="store", dest="disable",
                       metavar="uuid", help="disable a GNOME Shell extension")
    group.add_argument("-D", "--disable-all", dest="disableall",
                       action="store_true", help="disable all GNOME Shell extensions")
    group.add_argument("-e", "--enable", nargs=1, action="store", dest="enable",
                       metavar="uuid", help="enable a GNOME Shell extension")
    group.add_argument("-l", "--list-extension", dest="listext",
                       action="store_true", help="list GNOME Shell extensions")
    group.add_argument('-v', '--version', dest="listver",
                       action="store_true", help="list version numbers")
    args = parser.parse_args()
    if args.disable:
        disable_extension("".join(args.disable))
    elif args.disableall:
        disable_all_extensions()
    elif args.enable:
        enable_extension("".join(args.enable))
    elif args.listext:
        list_extensions()
    elif args.listver:
        list_versions()
    else:
        parser.print_usage()
        sys.exit(0)
if __name__ == "__main__":
    main()
```

Fairly major changes were required to make it work with GNOME 3.4. I do wish that the GNOME developers, most of whom appear to be young with little real practical world experience and who do not seem to understand the business need to preserve backwards compatability, would stop randomly changing APIs and functionality for no good reason. For example, Jasper St. Pierre removed support for tooltips in GNOME 3.4 in February 2012. See GNOME Bug 670034. His excuse:

> StTooltip has been plagued by lots of issues, and we recently ditched it in the dash. Remove it for good.

As a result, many of my GNOME Shell extensions require major rework to work with GNOME Shell 3.4. No consultation with end users, no attempt to fix the problems, no marking interfaces as deprecated for a release or two. Just remove the facility and let consumers of the facility rework their code. No wonder the GNOME Shell designers and developers are getting such a bad rap. By the way, look at the Cinnamon codebase, they fixed tooltips so that they work correctly.

This is the last release of the GNOME Shell that I will be releasing my extensions on. I simply do not have the time or inclination to rework my codebase everytime a young and usually inexperienced (in terms of years of software development) developer that is a member of the GNOME Shell cabel decides to reinvent facilities for no particularly good reason.

