GNOME Shell 3.6 DBus Support

Finnbarr P. Murphy
(fpm@fpmurphy.com)

I have written before about the DBus support in GNOME Shell. In GNOME Shell 3.6, there are significant changes to the DBus interface; hence this post.

The file that contains the DBus interface definitions is *shellDBus.js*. It contains the following two DBus interface definitions:

This is different than in previous versions of GNOME Shell which had only one Dbus interface.

For example, here is the GNOME Shell DBus interface definition from GNOME 3.4

Most of the changes make sense except for the spurious removal of *ApiVersion* which was removed by Jasper StPierre in July 2012. Of course, as usual, Jasper provided no rationale or advance warning for the removal of this DBus method. Typical GNOME cabal behaviour! By the way, 2012 was the year that GNOME users learned that their was no community around GNOME – there was simply the GNOME cabal and the unwashed rest.

Here is a small Python utility which uses the new DBus interface to list your installed extensions:

```
#!/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             4:"out of date",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"}
GNOME_SHELL_IFACE = 'org.gnome.Shell'
GNOME_SHELL_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,
                 GNOME_SHELL_IFACE, GNOME_SHELL_PATH, 'org.gnome.Shell.Extensions', 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 what the above Python script outputted on the system on which I am writing this post:

```
./example1.py
startofweek@fpmurphy.com: enabled, per user
poweroptions@fpmurphy.com: enabled, per user
removeusername@fpmurphy.com: initialized, per user
removeusermenuseparators@fpmurphy.com: enabled, per user
user-theme@gnome-shell-extensions.gcampax.github.com: enabled, system
disableactivitiesbutton@fpmurphy.com: initialized, per user
```

```
noa11y@fpmurphy.com: enabled, per user
weather@fpmurphy.com: enabled, per user
lookingglassbutton@fpmurphy.com: enabled, per user
```

The following simple Python utility using DBus to remove an installed extension:

```
#!/usr/bin/python
try:
    import os
    import sys
    import argparse
except ImportError, detail:
    print detail
    sys.exit(1)
from gi.repository import Gio, GLib
GNOME_SHELL_IFACE = 'org.gnome.Shell'
GNOME_SHELL_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,
                 GNOME_SHELL_IFACE, GNOME_SHELL_PATH, 'org.gnome.Shell.Extensions', None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
    def uninstall_extension(self, uuid):
        return self.proxy.UninstallExtension('(s)', uuid)
def main():
    parser = argparse.ArgumentParser(description="GNOME Shell uninstall tool")
    group = parser.add_mutually_exclusive_group()
    group.add_argument("-u", "--uninstall", nargs=1, action="store", dest="uninstall",
                       metavar="uuid", help="uninstall extension")
    args = parser.parse_args()
    if args.uninstall:
        s = ExtensionTool()
        s.uninstall_extension("".join(args.uninstall))
        print "%r is now uninstalled." % ("".join(args.uninstall))
    else:
        parser.print_usage()
        sys.exit(0)
if __name__ == "__main__":
    main()
```

and, yes, it really does remove all the files belonging to the extension!

Here is a simple Python utility which you can use to automatically download and install an extension from *http://extensions.gnome.org*:

```
#!/usr/bin/python
try:
    import os
    import sys
    import argparse
except ImportError, detail:
    print detail
    sys.exit(1)
from gi.repository import Gio, GLib
GNOME_SHELL_IFACE = 'org.gnome.Shell'
GNOME_SHELL_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,
                GNOME_SHELL_IFACE, GNOME_SHELL_PATH, 'org.gnome.Shell.Extensions', None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
    def install_extension(self, uuid):
        return self.proxy.InstallRemoteExtension('(s)', uuid)
def main():
    parser = argparse.ArgumentParser(description="GNOME Shell install tool")
    group = parser.add_mutually_exclusive_group()
    group.add_argument("-i", "--install", nargs=1, action="store", dest="install",
                       metavar="uuid", help="install extension")
    args = parser.parse_args()
    if args.install:
        s = ExtensionTool()
        s.install_extension("".join(args.install))
        print "%r is now installed." % ("".join(args.install))
    else:
        parser.print_usage()
        sys.exit(0)
if __name__ == "__main__":
    main()
```

You need to know the UUID of the extension that you want to install in order to use this utility. For example the UUID for the *hidetopbar* by
Mathieu Lutfy is *hidetopbar@mathieu.bidon.ca*. Mathieu based this extension, with my permission, on one of my early GNOME Shell extensions which hid the top panel when not required. Note this utility will fail to install the extension if *unzip* is not installed on your system or on your path as there is a builtin dependency on this utility in the GNOME Shell *extensionDownloader.js* source.

For my final example, this simple utility flashes an area of your screen. Why this method would ever be required by Gnome Shell, I have no idea.

```
#!/usr/bin/python
try:
    import os
    import sys
    import argparse
except ImportError, detail:
    print detail
    sys.exit(1)
from gi.repository import Gio, GLib
GNOME_SHELL_IFACE = 'org.gnome.Shell'
GNOME_SHELL_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,
                GNOME_SHELL_IFACE, GNOME_SHELL_PATH, 'org.gnome.Shell', None)
        except:
            print "Exception: %s" % sys.exec_info()[1]
    def flash_area(self, x, y, width, height):
        return self.proxy.FlashArea('(iiii)', x, y, width, height)
def main():
    parser = argparse.ArgumentParser(description="GNOME Shell flash tool")
    parser.add_argument("x", type=int)
    parser.add_argument("y", type=int)
    parser.add_argument("width", type=int)
    parser.add_argument("height", type=int)
    args = parser.parse_args()
    # hack - better argument checking should be provided
    if len(sys.argv) == 5:
```



```
        s = ExtensionTool()
        s.flash_area(args.x, args.y, args.width, args.height)
    else:
        parser.print_usage()
        sys.exit(0)
if __name__ == "__main__":
    main()
```

You need to pass in 4 arguments on this utility's command line, i.e *x*, *y*, *width* and *height*.

Other than the changes to the DBus interface, differences between Gnome Shell 3.4 and 3.6, as far as GNOME Shell extensions are concerned, are fairly easy to code around. Certainly they are far less significant to deal with than the changes that were required to port GNOME Shell extensions from 3.2 to 3.4.

