Intent-Type Content Sharing Between Applications With Dbus

Reasoning

Other platforms (especially mobile) provide a mechanism for sharing content and/or files between applications.

This is an intuitive and convenient way of distributing content.

Proposal Basics

Dbus could be used to handle intent registration and informing sharing applications of which receiving applications are available.

Each application would be responsible for registering any sharing intent with Dbus. This could be done at install or update, or event on application launch. The application should also be able to update the list of mime-types it shares (when a new version is installed, for example).

Applications could ask Dbus for a list of other Applications that would accept a given mime-type.

Dbus would keep the register in the file system (shared between User ans System level).

Applications could implement a simple Dbus interface to receive shared content.

Proposed Workflow

Application A (at launch or Install/Update) → (Register Accepted mime-types) → Dbus

Application B (at launch or Install/Update) → (Register Accepted mime-types) → Dbus (Dbus does not keep duplicates)

Application B User wants to share some content

Application B → (mime-type) → Dbus to get list of possible destination applications

Dbus → (list of Name and ID of applications) → Application B

User selects destination application Application A

Application B → (mime-type and base64 content string) → Application A

Application A → (ACK message w/status code) → Application B

Application A treats the content as designed…

Sample Minimal Implementation

Code available here: https://github.com/brainstormtrooper/gnome_sharing

DBus Service

DBus sharing service needs to implement two functions:

- get_sharing_apps(mime)
  - Param “mime” is a string mime-type (ex: text/plain)
  - Returns a list of declared sharing apps (see Registration Object Sample below)
- register_sharing(ob)
  - Param “ob” is a JSON string describing the accepted share offered by the application (see Registration Object Sample below)
  - Returns a JSON string
    - {“status_code”: 202, “message”: “Registration created”} if not already registered
    - {“status_code”: 200, “message”: “Already registered”} if already registered

Registration Object Sample

```
{
      'human_name': 'example app',
      'bus_name': 'com.example.GtkApplication',
      'share_types': [ 'text/plain' ],
      'share_files': True
 }
```

Functions Code Sample

```
    @dbus.service.method("com.brainstormtrooper.ShareService", in_signature='s',
out_signature='s')
    def get_sharing_apps(self, mime):
        res = []
        for s in self.subscribers:
            match = False
            for shareable in s['share_types']:
                if mime == shareable or (shareable.split('/')[1] == '*' and
shareable.split('/')[0] == mime.split('/')[0]):
                    match = True
            if match:
                res.append(s)
        return json.dumps(res)
```

```
    @dbus.service.method("com.brainstormtrooper.ShareService", in_signature='s',
out_signature='s')
    def register_sharing(self, ob):
        myob = json.loads(ob)
        if myob not in self.subscribers:
            self.subscribers.append(myob)
            print("New subscriber:", myob['human_name'])
            res = '{“status_code”: 202, “message”: “Registration created”}'
        else:
            res = '{“status_code”: 200, “message”: “Already registered”}'
        return res
```

Applications

Applications need to implement a sharing interface on Dbus to accept shared content:

The interface must accept and return a string (JSON).

Sample Interface Declaration

```
interface_xml = """
<node>
  <interface name='com.example.GtkApplicationSender.sharing'>
    <method name='share_content'>
      <arg type='s' name='content' direction='in'/>
      <arg type='s' name='response' direction='out'/>
    </method>
  </interface>
</node>
"""
Gio.DBusConnection.register_object(
    bus,
    '/com/example/GtkApplicationSender/sharing',
    Gio.DBusNodeInfo.new_for_xml(interface_xml).interfaces[0],
    self.do_handle_incoming_share
)
```

The application should implement a callback function to treat the received content:

- do_handle_incoming_share(content)
    - Param “content” is a JSON string containing the mime_type and base64 encoded content being shared.
        - {“mime_type”: “text/plain”, “contents”: “<<base64_encoded_string>>”}
    - Returns a JSON string containing a staus_code and a message.
        - {"status_code": 200, "message": "Accepted"}
        - {"status_code": 400, "message": "Invalid"}

Callback Sample

```
def do_handle_incoming_share(self, c, d, path, interface, method, payload,
invocation):
    print(payload)
    response = '{"status_code": 200, "message": "Accepted"}'
    self.label.set_label(payload.unpack()[0])
    invocation.return_value(GLib.Variant('(s)', (response,)))
```

The application should make two calls to the Dbus sharing service:

(See above Dbus service description for details of these functions)

- register_sharing(object): Informs Dbus of what content the calling application can share
- get_sharing_apps(mime_type): Asks Dbus for a list of applications that share a certain mime-type

Sample Calls to Dbus Service

```
bus = Gio.bus_get_sync(Gio.BusType.SESSION, None)
proxy = Gio.DBusProxy.new_sync(
    bus,
    Gio.DBusProxyFlags.NONE,
    None,
    'com.brainstormtrooper.ShareService',
    '/com/brainstormtrooper/ShareService',
    'com.brainstormtrooper.ShareService',
    None
)
reg = {
        'human_name': 'My app',
        'bus_name': 'com.example.GtkApplication',
        'share_types': [ 'text/plain' ],
        'share_files': True
    }
result = proxy.register_sharing('(s)', json.dumps(reg))

registered = proxy.get_sharing_apps('(s)', 'text/plain')
```

Finally, applications should implement a call to the receiving application on the bus.

(See share reception above for more information)

- share_something(content)
    - Param content is a JSON string conting the mime_type and base64 encoded content to share {“mime_type”: “text/plain”, “contents”: “<<base64_encoded_string>>”}

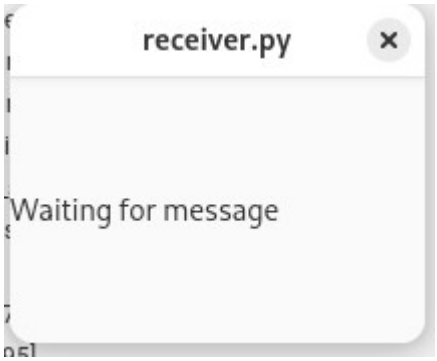
Sample Share Call

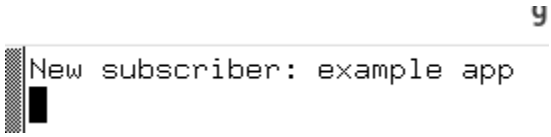
```
myproxy = Gio.DBusProxy.new_sync(
    bus,
    Gio.DBusProxyFlags.NONE,
    None,
```

```
            item.get_subtitle(),
            '/' + item.get_subtitle().replace('.','/')  + '/sharing',
            item.get_subtitle() + '.sharing',
            None
        )
        payload = {'mime_type': 'text/plain', 'contents': GLib.base64_encode(b'some
text')}
        res = myproxy.share_content('(s)', json.dumps(payload))
```

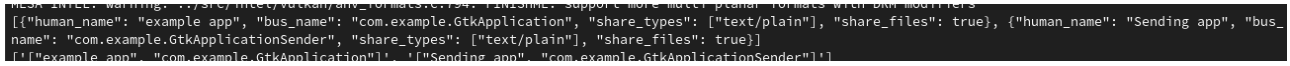
Demo

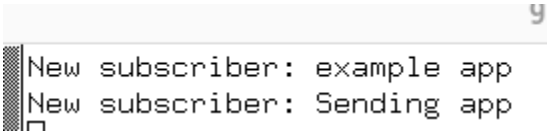
1. Launch the service: python3 ./service.py
2. Launch the receiving application: python3 ./receiver.py
   Note that the Application has registered itself with the Service and is waiting for content

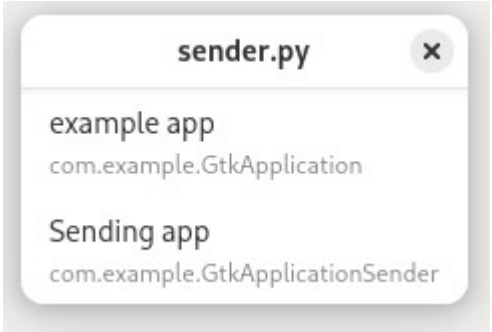




3. Launch the sending application: python3 ./sender.py

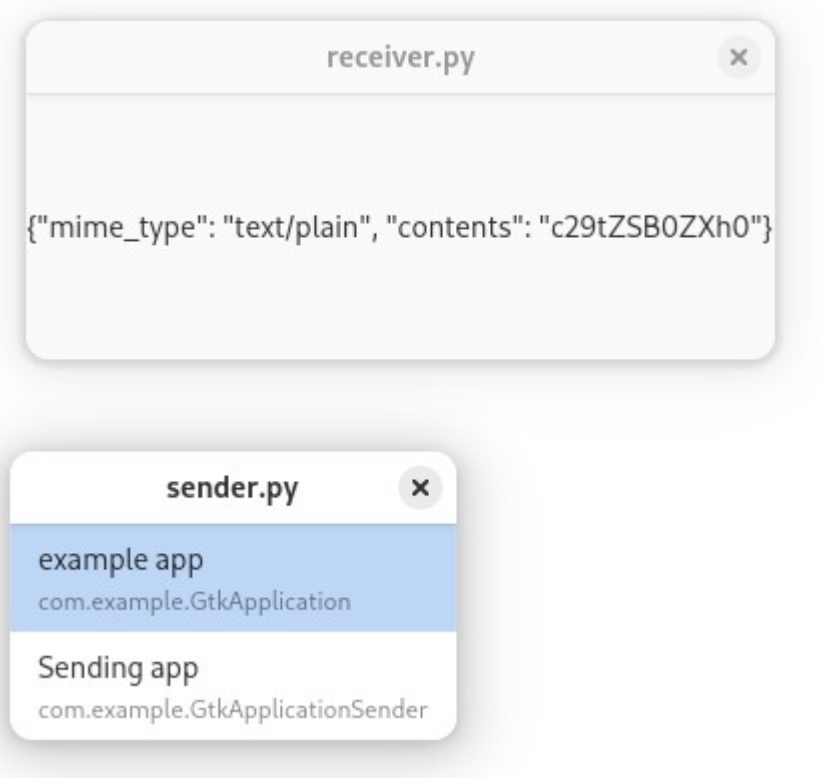






   The sending application also registers with the Service.
   It asks for a list of applications that accept text/plain. The Service responds and the Application displays the list of choices.

4. When the example app (receiver) is selected, the sending application makes a Dbus call to that application to send the content.



We see that the receiving application was sent a JSON payload with the mime-type and the contents base64 encoded. The receiving application responded with a success message.

```
com.example.GtkApplication
{"status_code": 200, "message": "Accepted"}
```